InfiniBand™ Architecture Specification Volume 1

Release 1.2

October 2004
Final Release

Table 1 Revision History

Revision	Release Date	
1.0	9/26/2000	Release 1.0
1.0.a	6/19/2001	Release 1.0 augmented with errata material. Updates only correct errors - no additional features have been added.
1.1	11/06/2002	Release 1.0.a augmented with additional features. Revised SA and CM Class with new version.
1.2	9/7/2004	Release 1.1 augmented with additional features (added annexes A7 through A10). Incorporated errata.

LEGAL DISCLAIMER

This IBTA specification provided "AS IS" and without any warranty of any kind, including, without limitation, any express or implied warranty of non-infringement, merchantability or fitness for a particular purpose.

In no event shall IBTA or any member of IBTA be liable for any direct, indirect, special, exemplary, punitive, or consequential damages, including, without limitation, lost profits, even if advised of the possibility of such damages.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

This is Volume 1 of the InfiniBand Architecture specification. It is the first in a series of documents that describe the architecture.

1.1 ACKNOWLEDGMENTS

The following persons were instrumental in the development of this volume of the InfiniBand Architecture specification:

Steering Committee Members

Co-chairs:

Tom Bradicich	Tom Macdonald	

Members:

Jacqueline Balfour	Ken Jansen	John Pescatore
Kevin Deierling	Michael Krause	Jim Pinkerton
Balint Fleischer	Todd Matters	Martin Whittaker
Dr. Alfred Hartmann	Ed Miller	Bob Zak
David Heisey		

Technical Working Group Members

Co-chairs:

Dwight Barron	Paul Grun	Jeff Hilland
Irv Robinson	David Wooten	

Members:

Dr. Alan Benner	Dr. Alfred Hartmann	Dr. Gregory F. Pfister
Mark Bradley	Michael Krause	Greg Still
Wolfgang Christl	Bill Lynn	Ken Ward
Diego Crupnicoff	Ed Miller	

Working Group Co-Chairs

Link Working Group (LWG):

Daniel Cassiday	Michael Krause

Software Working Group (SWG):

Ed Miller	Renato J. Recio	Jim Pinkerton

Management Working Group (MgtWG):

David W. Abmayr	Brian Forbes	Jeff Hilland
Dr. Pankaj Mehra	Dr. Gregory F. Pfister	William H. Swortwood
Dr. Mazin Yousif		

Application Working Group (AWG):

Dwight Barron	William Futral	Greg Pellegrino

Contributors

David W. Abmayr
Dr. Ramon Acosta
Sesidhar Baddela
Dr. Alan Benner
Frank L. Berry
Bruce Beukema
Suri Brahmaroutu
David M. Brean
Tom Brey
John Carrier
Daniel Cassiday
Norman Chou
Ian Colloff
Joe Cowan
David Craddock
Olivier Crémel
Diego Crupnicoff
Paul Culley
Roger Cummings
George Dake
Ellen Deleganes
Kevin Deierling
Chunlei Dong
Scott Feller
Brian Forbes
Dan Fowler
Giles Frazier
Bill Futral
Dr. Freddy Gabbay
Narayanan Ganapathy
David Garcia
Nimrod Gindi
Dror Goldenberg
Nancy J. Golio
Paul Grun
James Hamrick
Dr. Alfred Hartmann
Yaron Haviv
Arel Hendel
Jeff Hilland
Michael Heumann
Jenwei Hsieh
Christopher J. Jackson
Jeff Jilg
Dave Kasberg
Vivek Kashyap
Dr. Ted Kim
Dr. Hiro Kishimoto
Michael Krause
Alan Langerman
James W. Livingston
Venitha L. Manter
Gunna Marripudi
Dr. Pankaj Mehra
Charles Monia
Jim Mott
Mark Myers
Neil MacLean
Tarl Neustaedter
Rahul Nim
Shravan Pargal
Joe Pelissier
Greg Pellegrino
Dr. Gregory F. Pfister
Jim Pinkerton
Vandana Rao
Renato J. Recio
Roger Ronald
William J. Rooney
Hal Rosenstock
Tom Ryle
Hide Senta
Michael Shinkarovsky
Cris Simpson
William Strahm
William H. Swortwood
Chris Szeto
Monika ten Bruggencate
Pat Thaler
Saeki Toshiaki
Franco Travastino
Dono Van-Mierop
Ken Ward
Kurt Ware
Tom Webber
Dong Wei
Jeff Young
Dr. Mazin Yousif

1.2 INFINIBAND CONCEPTUAL OVERVIEW

The InfiniBand Architecture Specification describes a first order interconnect technology for interconnecting processor nodes and I/O nodes to form a system area network. The architecture is independent of the host operating system (OS) and processor platform.

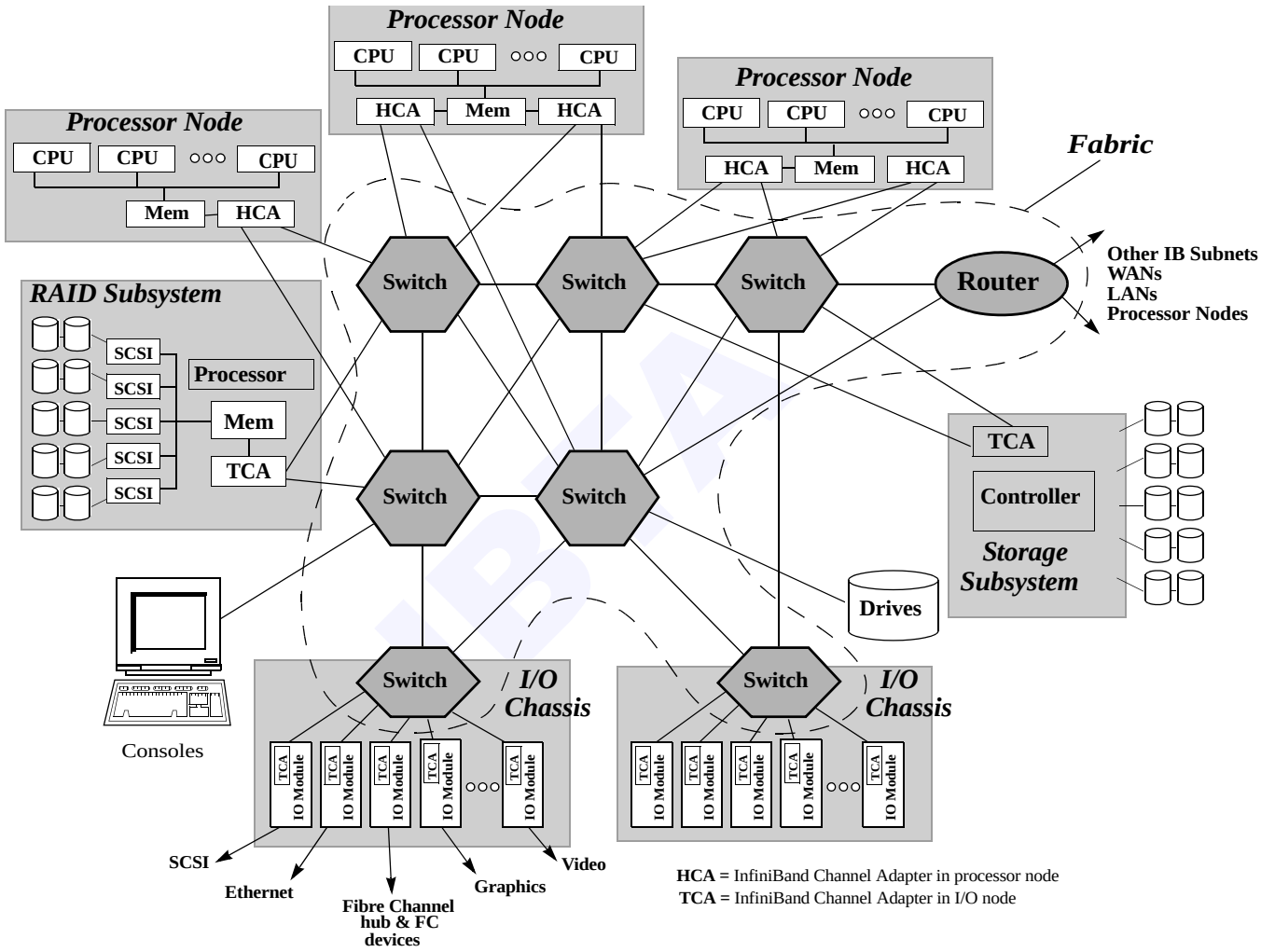


Figure 1 IBA System Area Network

1.2.1 THE PROBLEM

Existing interconnect technologies have failed to keep pace with computer evolution and the increased burden imposed on data servers, application processing, and enterprise computing created by the popular success of the internet. High-end computing concepts such as clustering, fail-safe, and 24x7 availability demand greater capacity to move data between processing nodes as well as between a processor node and I/O devices. These trends require higher bandwidth and lower latencies, they are

pushing more functionality down to the I/O device, and they are demanding greater protection, higher isolation, deterministic behavior, and a higher quality of service than currently available.

1.2.2 FEATURES

InfiniBand Architecture (IBA) is designed around a point-to-point, switched I/O fabric, whereby end node devices (which can range from very inexpensive I/O devices like single chip SCSI or ethernet adapters to very complex host computers) are interconnected by cascaded switch devices. The physical properties of the IBA interconnect support two predominant environments, with bandwidth, distance and cost optimizations appropriate for these environments:

- Module-to-module, as typified by computer systems that support I/O module add-in slots
- Chassis-to-chassis, as typified by interconnecting computers, external storage systems, and external LAN/WAN access devices (such as switches, hubs, and routers) in a data-center environment.

The IBA switched fabric provides a reliable transport mechanism where messages are enqueued for delivery between end nodes. In general, message content and meaning is not specified by InfiniBand Architecture, but rather is left to the designers of end node devices and the processes that are hosted on end node devices. IBA defines hardware transport protocols sufficient to support both reliable messaging (send/receive) and memory manipulation semantics (e.g. remote DMA) without software intervention in the data movement path. IBA defines protection and error detection mechanisms that permit IBA transactions to originate and terminate from either privileged kernel mode (to support legacy I/O and communication needs) or user space (to support emerging interprocess communication demands).

The IBA Specification also addresses the need for a rich manageability infrastructure to support interoperability between multiple generations of IBA components from many vendors over time. This infrastructure provides ease of use and consistent behavior for high volume, cost sensitive deployment environments. IBA also specifies interfaces for industry standard management that interoperate with enterprise class management tools for configuration, asset management, error reporting, performance metric collection, and topology management necessary for data center deployment of IBA.

1.2.3 BENEFITS

For all of the revolutionary aspects of IBA, the architecture has been carefully designed to minimize disruption of prevailing market paradigms and business practices. By simultaneously supporting board and chassis in-

terconnections, it is expected that vendors are able to adopt InfiniBand Architecture technology for use in future generations of existing products, within current business practices, to best support their customers needs.

IBA can support bandwidths that are anticipated to remain an order of magnitude greater than prevailing I/O media (SCSI, Fibre Channel, Ethernet). This ensures its role as the common interconnect for attaching I/O media using these technologies. Reinforcing this point is IBA's native use of IPv6 headers, which supports extremely efficient junctions between IBA fabrics and traditional internet and intranet infrastructures.

IBA supports implementations as simple as a single computer system, and can be expanded to include: replication of components for increased system reliability, cascaded switched fabric components, additional I/O units for scalable I/O capacity and performance, additional host node computing elements for scalable computing, or any combinations thereof. InfiniBand Architecture is a revolutionary architecture that enables computer systems to keep up with the ever increasing customer requirement for increased scalability, increased bandwidth, decreased CPU utilization, high availability, high isolation, and support for Internet technology.

Being designed as a first order network, IBA focuses on moving data in and out of a node's memory and is optimized for separate control and memory interfaces. This permits hardware to be closely coupled or even integrated with the node's memory complex, removing any performance barriers. IBA is flexible enough to be implemented as a second order network permitting legacy and migration. Even when implemented as a second order network, IBA's memory optimization operation permits maximum available bandwidth utilization and increases CPU efficiency.

1.3 SCOPE

IBA supports a range of applications from being the backplane interconnect of a single host, to a complex system area network consisting of multiple independent and clustered hosts and I/O components.

For the single host environments, as depicted in Figure 2, each IBA fabric serves as a private I/O interconnect for its host and provides connectivity

between the host's CPU/memory complex and a number of I/O modules. For this environment, all devices are dedicated to the host.

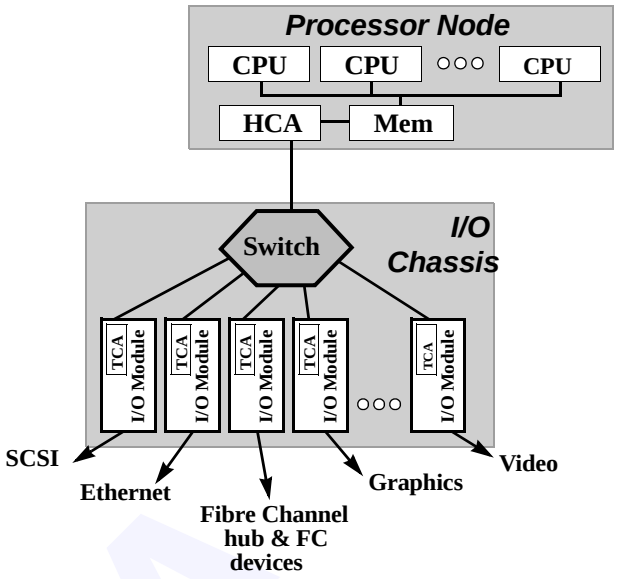


Figure 2 Single Host Environment

On the other end of the scale is multiple host connectivity as depicted in Figure 1. Here a single fabric or even multiple fabrics interconnect numerous hosts and various I/O units. Some hosts might share I/O devices and others do not. Interprocess communication between hosts becomes a very significant objective. Trivial fabric management is no longer sufficient as network administrators desire additional features to maintain separation and assure deterministic behavior.

The architecture not only specifies the mechanisms for I/O and interprocess communication, but it also specifies an extensive set of management mechanisms that are flexible enough to permit single host environments with out undue burden and costly fabric managers and at the same time support very complex system area networks (SAN) and feature rich fabric management.

1.4 DOCUMENT ORGANIZATION

1.4.1 SERIES OF VOLUMES

There are two volumes that comprise the InfiniBand normative specifications suite:

Volume 1 - specifies the core InfiniBand™ Architecture. It provides normative information required for IBA operation for switches, routers, host channel adapters for processor nodes, target channel adapters for I/O devices, and management.

Volume 2 - specifies electrical & mechanical configurations. It specifies requirements for a number of different physical media and signaling rates, defines mechanical form factors, and specifies physical and chassis management requirements.

1.4.2 VOLUME 1 ORGANIZATION

1.5 DOCUMENT CONVENTIONS

1.5.1 BYTE ORDERING

This specification uses Big Endian byte ordering. For fields greater than one byte in size this means that the most significant byte of each field is transmitted first as illustrated in Figure 3.

Byte offset	previous field
+0	Most Significant Byte
+1	Second Most Significant Byte
.	o
.	o
.	o
+n	Least Significant Byte

Figure 3 Byte Order for Multiple Byte Fields

Unless specifically stated otherwise, the text of this document lists fields in the order of transmission. In most cases, multiple byte fields are aligned to start or end on a 32-bit boundary. For clarity, certain figures show fields ordered in 32 bit words. These words are in big endian format and implementations targeted for little endian processing need to pay particular attention to byte ordering to assure correct operation since little endian processing tends to place the least significant bytes in lower byte offsets.

Figure 4 illustrates how numeric and bit significant fields should be interpreted.

bits	b7 b0	b7 b0	b7 b0	b7 b0
Byte Offset	Byte 0,4,8.	Byte 1,5,9,...	Byte 2,6,10,...	Byte 3,7,11,...
0-3	b15 16-bit field b0		b15 16-bit field b0	
4-7	b31 32-bit field b0			
8-11	b7 1-byte b0	b23 24-bit field b0		
12-15	b23 24-bit field b0		b7 1-byte b0	
16-19	b47 48-bit field (high) b16			
20-23	b15 48-bit field (low) b0		b47 48-bit field (high bytes) b32	
24-27	b31 48-bit field (low bytes) b0			
28-31	b63 64-bit field (high bytes) b32			
32-35	b31 64-bit field (low bytes) b0			
36-39	b127 128-bit field (highest bytes) b96			
40-43	b95 b64			
44-47	b63 b32			
48-51	b31 128-bit field (lowest bytes) b0			

Figure 4 Byte Order Examples

Bit fields with other than byte granularity follow the same rules - that is, the most significant bits of the field occupies the higher order bits of the lowest byte offset with least significant bits being in the lowest byte offset as illustrated in Figure 5.

Previous Byte	First Byte								Next Byte								Following Byte
	5-bit field					3-bit field			2-bit		6-bit field						
	b4	b3	b2	b1	b0	b2	b1	b0	b1	b0	b5	b4	b3	b2	b1	b0	
	4-bit field				12-bit field												
	b3	b2	b1	b0	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0	
	14-bit field														2-bit		
	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0	b1	b0	

Figure 5 Bit Order Examples

1.5.2 NUMERIC VALUES

Unless otherwise stated numerical values without qualifiers are decimal. This document uses the following qualifiers:

- 0x prefixed to a hexadecimal value (e.g., 0x15F7)
- b’ prefixed to a binary value (e.g., b’0110)

An obvious exception are binary numbers used in figures and tables

In table headings a colon is used to specify a range of bits (e.g. Bits 7:0) and table values in that column are binary numbers.

A dash between two numbers represents a range (e.g. 0-3 = zero to three)

Global IDs are 128-bit values specified in the format :

value:value:value:value:value:value:value:value

Where each value represents a 4-digit hexadecimal number (e.g., FF02:0:0:0:0:0:0:1)

1.6 DISCLAIMER

Like any document, this specification is subject to errata for correctness, clarity, and enhancements. The InfiniBand℠ Trade Association hosts a web site at http://www.InfiniBandTA.org. Please visit this site to check for errata and updates to this specification.

CHAPTER 2: GLOSSARY

Active
Describes an entity initiating a communication establishment request (e.g., TCP CONNECT).

Address Handle
An object that contains the information necessary to transmit messages to a remote port over Unreliable Datagram service.

Address Vector
A collection of address and Path information specifying a remote port and the parameters to be used when communicating with it.

AETH
Ack Extended Transport Header

AM
Attribute Modifier.

Asynchronous error
A permanent error that cannot be reported through immediate or completion error handling mechanisms at the local end. Asynchronous errors may be unaffiliated or may be affiliated with a specific Completion Queue, Endport, or Queue Pair.

Attribute
The collection of management data carried in a Management Datagram.

Automatic Path Migration
The process in which a Channel Adapter, on a per-Queue Pair basis, signals another CA to cause Path Migration to a preset alternate Path. Automatic Path Migration uses a bit in a request or response packet (MigReq) to signal the other channel adapter to migrate to the predefined alternate path.

B_Key
See Baseboard Management Key.

Base LID
The numerically lowest Local Identifier that refers to a Port. The Path Bits of a Base LID are always zero.

Base Switch Port 0
A Switch Port 0 which is not an Endport.

Baseboard Managed Unit
Any Unit which provides InfiniBand(TM) specification defined information about itself by a Baseboard method MAD operation through the InfiniBand™ link.

Baseboard Management Key
A construct that is contained in IBA management datagrams to authenticate that the sender is allowed to perform the requested operation.

Binding
The act of associating a virtual address range in a specified Memory Region with a Memory Window.

BTH Base Transport Header.

CA See Channel Adapter.

Channel The association of two queue pairs for communication.

Channel Adapter Device that terminates a link and executes transport-level functions. One of Host Channel Adapter or Target Channel Adapter.

Channel Interface The presentation of the channel to the Verbs Consumer as implemented through the combination of the Host Channel Adapter, associated firmware, and device driver software.

Channel, Reliable Datagram See Reliable Datagram Channel.

CI See Channel Interface.

Client The active entity in an active/passive communication establishment exchange.

CM See Communication Manager.

CME Chassis Management Entity.

Communication Manager The software, hardware, or combination of the two that supports the communication management mechanisms and protocols.

Completion Error Permanent interface or processing error reported through completion status.

Completion Event Handler A handler that is invoked when the Consumer requests completion notification and an entry is added to the completion queue associated with the handler's identifier.

Completion Queue A queue containing one or more Completion Queue Entries. Completion Queues are internal to the Channel Interface, and are not visible to verb consumers.

Completion Queue Entry The Channel Interface-internal representation of a Work Completion.

Component Mask A field in a Management Datagram used to indicate which components of the MAD are to be considered in carrying out the operation. See Wildcarding.

Connection An association between a pair of entities (e.g., processes) over one or more Channels.

Consumer See Verbs Consumer.

CQE	Completion Queue Entry, commonly pronounced “cookie”.
CRC	Cyclic Redundancy Check.
Data Payload	The data, not including any control or header information, carried in one packet.
Data Segment	A tuple in a Work Request that specifies a virtually contiguous buffer for Host Channel Adapter access. Each Data Segment consists of a Virtual Address, an associated Local Key or Remote Key, and a length.
DETH	Datagram Extended Transport Header.
DGID	Destination Global Identifier.
DLID	Destination Local Identifier.
EEC	See End to End Context.
EECN	See End to End Context Number.
EE Context	See End to End Context.
Endport	A Port which can be a destination of LID-routed communication within the same Subnet as the sender. All Channel Adapter ports on the subnet are endports of that subnet, as is Port 0 of each Switch in the subnet. Switch ports other than Port 0 may not be endports. When *port* is used without qualification, it may be assumed to mean *endport* whenever the context indicates that it is a destination of communication.
End to End Context	The endpoint of a Reliable Datagram channel.
End to End Context Number	Identifies a specific End to End Context within a Channel Adapter.
End to End Flow Control	A mechanism to prevent a sender from transmitting messages during periods when receive buffers are not posted at the recipient.
Enhanced Switch Port 0	A Switch Port 0 which provides the functionality of a Target Channel Adapter.
External Switch Port	A physical Port on a Switch. See also Switch Port 0.
Fabric	The collection of Links, Switches, and Routers that connects a set of Channel Adapters.
Fast Register Physical MR	A memory registration performed on an existing local L_Key, and any associated R_Key, through a Post Send Work Request.

Fast Register PMR — Fast Register Physical MR.

Gb/s — Giga-bits per second (10^9 bits per second)

GB/s — Giga-bytes per second (10^9 bytes per second)

General Service Interface — An interface providing management services (e.g., connection, performance, diagnostics) other than subnet management.

GID — See Global Identifier.

GID Index 0 — The unicast GID referenced through index 0 of an endport's GID Table, based on the endport's invariant manufacturer-assigned EUI-64.

GID Table — A table containing one or more Global Identifiers by which an endport may be referenced.

Global Identifier — A 128-bit identifier used to identify an Endport or a multicast group. GIDs are valid 128-bit IPv6 addresses (per RFC 2373) with additional properties / restrictions defined within IBA to facilitate efficient discovery, communication, and routing.

Global Route Header — Routing header present in InfiniBand™ Architecture packets targeted to destinations outside the sender's local subnet.

Globally Unique Identifier — A number that uniquely identifies a device or component.

GMP — General Management Packet.

GRH — See Global Route Header.

GSI — See General Service Interface.

GUID — See Globally Unique Identifier.

HCA — See Host Channel Adapter.

Host — One or more Host Channel Adapters governed by a single memory/CPU complex.

Host Channel Adapter — A Channel Adapter that supports the Verbs interface.

IBA — InfiniBand™ Architecture.

IB-ML — InfiniBand™ Management Link.

ICRC — See Invariant CRC.

Immediate Data

Data contained in a Work Queue Element that is sent along with the payload to the remote Channel Adapter and placed in a Receive Work Completion.

Immediate Error

A permanent Interface Error reported through the verb status.

Initiator

The source of requests.

Interface Error

An error due to an invalid field in a Work Request.

Invalid Key

See Key.

Invalidate Operation

An operation that disables CI's access to host memory through an L_Key or R_Key, but retains L_Key or R_Key translation and protection resources for use on future memory registrations.

Invariant CRC

A CRC covering the fields in a packet that do not change from the source to the destination.

I/O

Input/Output.

I/O Controller

One of the two architectural divisions of an I/O Unit. An I/O controller (IOC) provides I/O services, while a Target Channel Adapter provides transport services.

I/O Unit

An I/O unit (IOU) provides I/O service(s). An I/O unit consists of one or more I/O Controllers attached to the fabric through a single Target Channel Adapter.

I/O Virtual Address

An address having no direct meaning to the Host processor, intended for use only in describing a Local or Remote memory buffer to the Host Channel Adapter.

IOC

See I/O Controller.

IOU

See I/O Unit.

IPv6

Internet Protocol, version 6

IPv6 Address

A 128-bit address constructed in accordance with IETF RFC 2460 for IPv6.

Key

A construct used to limit access to one or more resources, similar to a password. The following keys are defined by the InfiniBandTM Architecture:

Baseboard Management Key

Local Key

Management Key

Queue Key

Partition Key

Remote Key

Key Space Refers to whether the index portion of the L_Key and R_Key for a particular Memory Region have the same value or not. If they do, then they are in the same Key Space. Otherwise they are in separate Key Spaces.

L_Key See Local Key.

LID See Local Identifier.

LID Mask Control A per-port value assigned by the Subnet Manager. The value of the LMC specifies the number of Path Bits in the Local Identifier.

Link A full duplex transmission path between any two network fabric elements, such as Channel Adapters or Switches.

LMC See LID Mask Control.

Local Identifier An address assigned to a port by the Subnet Manager, unique within the subnet, used for directing packets within the subnet. The Source and Destination LIDs are present in the Local Route Header. A Local Identifier is formed by the sum of the Base LID and the value of the Path Bits.

Local Invalidate An Invalidate Operation performed through the Send Queue: on a local L_Key, R_Key, or Memory Region Handle that is associated with an MR; or an R_Key or Memory Window Handle that is associated with a Type 2 Memory Window.

Local Key An opaque object, created by a verb, referring to a Memory Registration, used with a Virtual Address to describe authorization for the HCA hardware to access local memory. It may also be used by the HCA hardware to identify the appropriate page tables for use in translating virtual to physical addresses.

Local Route Header Routing header present in all InfiniBand™ Architecture packets, used for routing through switches within a subnet.

Local Subnet The collection of links and Switches that connect the Channel Adapters of a particular subnet.

LRH See Local Route Header.

M_Key — See Management Key.

MAD — See Management Datagram.

Managed Unit — A Unit which provides Vital Product Data about itself to an external entity, and is managed by that entity.

Management Datagram — Refers to the contents of an Unreliable Datagram packet used for communication among HCAs, switches, routers, and TCAs to manage the fabric. InfiniBand™ Architecture describes the format of a number of these management commands.

Management Key — A construct that is contained in IBA management datagrams to authenticate the sender to the receiver.

Maximum Transfer Unit — The maximum Packet Payload size, which may be 256, 512, 1024, 2048, or 4096 bytes. See also MTU Capacity, Neighbor MTU, and Path Maximum Transfer Unit.

MB/s — Mega-bytes per second (10^6 bytes per second)

Memory Protection Attributes — The access rights granted to Memory Regions.

Memory Region — A virtually contiguous area of arbitrary size within a Consumer's address space that has been registered, enabling HCA local access and optional remote access. See Memory Registration

Memory Region Handle — An opaque object returned to the consumer when the consumer registers a Memory Region. The Memory Region Handle is used to specify the registered region to the memory management verbs.

Memory Registration — The act of registering a host Memory Region for use by a consumer. The memory registration operation returns a Memory Region Handle. The process provides this with any reference to a virtual address within the memory region.

Memory Window — An allocated resource that enables remote access after being bound to a specified area within an existing Memory Region. Each Memory Window has an associated Window Handle, set of access privileges, and current R_Key.

Message — A transfer of information between two or more Channel Adapters that consists of one or more packets.

Message-Level Flow Control — See End to End Flow Control.

Message Sequence Number — A value returned as part of an acknowledgement by the responder to the requestor, indicating the last message completed. Contrast Packet Sequence Number.

Modifiers — In a verb definition, the list of input and output objects that specify how, and on what, the verb is to be executed.

MR — Memory Region

MSN — See Message Sequence Number.

MTU — See Maximum Transfer Unit.

MTUCap — See MTU Capacity.

MTU Capacity — The largest Maximum Transfer Unit that a port can support.

Multicast — A facility by which a packet sent to a single address may be delivered to multiple ports.

Multicast Identifier — A Local Identifier or Global Identifier for a Multicast Group.

Multicast Group — A collection of Endports that receive Multicast packets sent to a single address.

MW — Memory Window

Neighbor MTU — The configured Maximum Transfer Unit for a Port, the value that specifies the maximum packet payload that may be sent to, or received from, the port at the other end of the Link.

NQ — Notification Queue.

Out-of-band Management — Management messages which traverse a transport other than the InfiniBand™ fabric.

Outstanding

1) The state of a Work Request after it has been posted on a Work Queue, but before the retrieval of the Work Completion by the consumer.
2) The state of a packet that has been sent onto the fabric but has not been acknowledged.

P_Key — See Partition Key.

Packet — The indivisible unit of IBA data transfer and routing, consisting of one or more headers, a Packet Payload, and one or two CRCs.

Packet Payload — The portion of a Packet between (not including) any Transport header(s)

and the CRCs at the end of each packet. The packet payload contains up to 4096 bytes.

Packet Sequence Number

A value carried in the Base Transport Header that allows the detection and re-sending of lost packets.

Partition

A collection of Channel Adapter ports that are allowed to communicate with one another. Ports may be members of multiple partitions simultaneously. Ports in different partitions are unaware of each other's presence insofar as possible.

Partition Key

A value carried in packets and stored in Channel Adapters that is used to determine membership in a partition.

Default Partition Key: A partition key special value providing Full membership in the default partition. See Partition Membership Type.

Invalid Partition Key: A special value that indicates that the Partition Key Table entry does not contain a valid key.

Partition Key Table

A table of partition keys present in each Port.

Partition Key Table Index (P_Key_ix)

An index into the partition key table.

Partition Manager

The entity that manages partition keys and membership.

Partition Membership Type

The high-order bit of the partition key is used to record the type of membership in an Port's partition table: 0 for Limited, 1 for Full. Limited members cannot accept information from other Limited members, but communication is allowed between every other combination of membership types.

Passive

Describes an entity waiting to receive a communication establishment request (e.g., TCP LISTEN).

Path

The collection of links, switches, and routers a message traverses from a source Channel Adapter to a destination channel adapter. Within a subnet, a path is defined by the tuple <SLID, DLID, SL>.

Path Bits

The portion of a Local Identifier that may be changed to vary the Path through the subnet to a particular Port. If the Path Bits are zero, the Local Identifier is equal to the Base LID. The number of Path Bits applicable to a particular port is specified by the Subnet Manager through the LID Mask Control value.

Path Maximum Transfer Unit

The Maximum Transfer Unit supported along a Path from source to destination. PMTU is described in terms of the payload size, and may be

256, 512, 1024, 2048, or 4096 bytes.

Path Migration — The modification of the Path used by a connection.

PD — See Protection Domain.

Peer — 1) One of the agents in an active/active connection establishment exchange.
2) A generic term for the entity at the other end of a connection.

Pinning memory — A function supplied by the OS which forces the memory region to be resident and keeps the virtual-to-physical translations constant from the HCA point of view.

PM — See Partition Manager.

PMR — Physical Memory Region

PMTU — See Path Maximum Transfer Unit.

Port — Location on a Channel Adapter or Switch to which a link connects. There may be multiple ports on a single Channel Adapter, each with different context information that must be maintained. Switches/switch elements contain more than one port by definition.

Post — To place a Work Request on a Work Queue.

Private Data — A field present in Communication Management messages that is opaque at all IBA layers. Consumers may use this field to “piggy-back” additional information over the CM message exchange.

Processing Error — A processing error is an error that occurs when the Host Channel Adapter is performing the unit of work described by the Work Queue Element and is unable to complete the request successfully due to an error that is returned by the transport protocol.

Protection Domain — A mechanism for associating Queue Pairs, Address Handles, Memory Windows, and Memory Regions.

PSN — See Packet Sequence Number.

Q_Key — See Queue Key.

QoS — See Quality of Service.

QP — See Queue Pair.

Quality of Service — Metrics that predict the behavior, reliability, speed, and latency of a given

network connection.

Queue Key — A construct that is used to validate a remote sender's right to access a local Receive Queue for the Unreliable Datagram and Reliable Datagram service types. If the Q_Key present in an incoming packet does not match the value stored in the receiving QP, the packet shall be dropped.

Queue Pair — Consists of a Send Work Queue and a Receive Work Queue. Send and receive queues are always created as a pair and remain that way throughout their lifetime. A Queue Pair is identified by its Queue Pair Number.

Queue Pair Context — The information that pertains to a particular Queue Pair, such as the current Work Queue Elements, Packet Sequence Numbers, transmission parameters, etc.

Queue Pair Handle — An opaque object that refers to a specific Queue Pair. A Queue Pair Handle is returned by the operation that creates the QP and is supplied as an identifying parameter for other QP operations.

Queue Pair Number — Identifies a specific Queue Pair within a Channel Adapter.

R_Key — See Remote Key.

Raw Datagram — A packet that contains an IBA Local Route Header, may contain an IBA Global Route Header, but does not contain an IBA Transport header, and is not handled by IBA transport services.

RC — See Reliable Connection.

RD — See Reliable Datagram.

RDC — See Reliable Datagram Channel.

RDD — See Reliable Datagram Domain.

RDETH — Reliable Datagram Extended Transport Header.

RDMA — See Remote Direct Memory Access.

Receive Queue — One of the two queues associated with a Queue Pair. The receive queue contains Work Queue Elements that describe where to place incoming data.

Region Handle — See Memory Region Handle.

Registered Memory — A region of memory that has been through Memory Registration.

Registration See Memory Registration.

Registered memory region See Memory Region.

Reliable Connection A Transport Service Type in which a Queue Pair is associated with only one other QP, such that messages transmitted by the send queue of one QP are reliably delivered to receive queue of the other QP. As such, each QP is said to be “connected” to the opposite QP.

Reliable Datagram A Transport Service Type in which a Queue Pair may communicate with multiple other QPs over a Reliable Datagram Channel. A message transmitted by an RD QP’s send queue will be reliably delivered to the receive queue of the QP specified in the associated Work Request. Despite the name, Reliable Datagram messages are not limited to a single packet.

Reliable Datagram Channel The association of two Reliable Datagram End to End Contexts. A Reliable Datagram channel may multiplex Reliable Datagrams from many RD Queue Pairs.

Reliable Datagram Domain An association that defines which Reliable Datagram Queue Pairs may use an End to End Context.

Reliable Multi-Packet Protocol A transaction protocol based on Management Datagrams supporting the reliable transfer of amounts of data larger than that possible in a single Management Datagram.

Remote Direct Memory Access Method of accessing memory on a remote system without interrupting the processing of the CPU(s) on that system.

Remote Invalidate An Invalidate Operation performed on a local R_Key through an incoming Send with Invalidate Message.

Remote Key An opaque object, created by a verb, referring to a Memory Region or Memory Window, used with a Virtual Address to describe authorization for the remote device to access local memory. It may also be used by the HCA hardware to identify the appropriate page tables for use in translating virtual to physical addresses.

Reserved L_Key An L_Key that can be used by a privileged Consumer to provide the HCA direct access to host physical addresses.

Retired The state of a Work Queue Element after the Host Channel Adapter completes the operation specified by the WQE, but before the Work Completion has been presented to the consumer.

RMPP See Reliable Multi-Packet Protocol.

RNR Nak — Receiver Not Ready. A response signifying that the receiver is not currently able to accept the request, but may be able to do so in the future.

Router — A device that transports packets between IBA subnets.

SA — See Subnet Administration.

SAR — Segmentation and Re-assembly.

Send Queue — One of the two queues of a Queue Pair. The Send queue contains WQEs that describe the data to be transmitted.

Server

1) The passive entity in a connection establishment exchange.
2) An entity (e.g., a process) that provides services in response to requests from clients.

Service ID — A value that allows a Communication Manager to associate an incoming connection request with the entity providing the service. The Service ID is similar to the TCP Port Number.

Service Level — Value in the Local Route Header identifying the appropriate Virtual Lane for a packet, enabling the implementation of differentiated services. While the appropriate VL for a specific Service Level may differ over a packet's Path, the Service Level remains constant.

Service Type — See Transport Service Type.

Signaled Completion — A modifier used for Work Requests submitted to the Send Queue specifying that a Work Completion shall be generated when the work requested completes, whether successfully or in error.

SGID — Source Global Identifier.

Shared Receive Queue — The Shared Receive Queue contains WQEs that can be used to receive incoming data on any QP that is associated with the Shared Receive Queue.

SLID — Source Local Identifier

SL — See Service Level.

SM — See Subnet Manager.

SMA — See Subnet Management Agent.

SMP — See Subnet Management Packet.

Solicited Event
A facility by which a message sender may cause an event to be generated at the recipient when the message is received.

SRQ
Shared Receive Queue

Subnet
A set of InfiniBand™ Architecture Ports, and associated links, that have a common Subnet ID and are managed by a common Subnet Manager. Subnets may be connected to each other through routers.

Subnet Administration
The architectural construct that implements the interface for querying and manipulating subnet management data.

Subnet Manager
One of several entities involved in the configuration and control of the subnet.

Master Subnet Manager: The subnet manager that is authoritative, that has the reference configuration information for the subnet.

Standby Subnet Manager: A subnet manager that is currently quiescent, and not in the role of a master SM, by agency of the master SM. Standby SMs are dormant managers.

Subnet Management Agent
An entity present in all IBA Channel Adapters and Switches that processes Subnet Management Packets from Subnet Manager(s).

Subnet Management Data
Vital Product Data required by the Subnet Manager.

Subnet Management Packet
The subclass of Management Datagrams used to manage the subnet. SMPs travel exclusively over Virtual Lane 15 and are addressed exclusively to Queue Pair Number 0.

Switch
A device that routes packets from one link to another of the same Subnet, using the Destination Local Identifier field in the Local Route Header.

Switch Management Port
A virtual port by which a Switch may be managed. See Switch Port 0.

Switch Port 0
An addressable virtual port by which a Switch may be managed. May be one of Base Switch Port 0 or Endport.

TCA
See Target Channel Adapter.

Target Channel Adapter
A Channel Adapter typically used to support I/O devices. TCAs are not required to support the Verbs interface. See also I/O Unit.

Transport Service Type
Describes the reliability, sequencing, message size, and operation types that will be used between the communicating Channel Adapters.

Transport service types that use the IBA transport:

- Reliable Connection
- Unreliable Connection
- Reliable Datagram
- Unreliable Datagram

Raw Datagram service does not use the IBA transport.

Type 1 Memory Window — A Memory Window that is associated with a QP through the PD and cannot be Invalidated.

Type 2 Memory Window — A Memory Window that is either a Type 2A Memory Window or a Type 2B Memory Window.

Type 2A Memory Window — A Memory Window that when bound is associated with a QP through the QP Number and can be Invalidated.

Type 2B Memory Window — A Memory Window that when bound is associated with a QP through the QP Number and PD, and can be Invalidated.

UC — See Unreliable Connection.

UD — See Unreliable Datagram.

Unicast — An identifier for a single port. A packet sent to a unicast address is delivered to the port identified by that address.

Unit — One or more sets of processes and/or functions attached to the fabric by one or more channel adapters. See Host and I/O Unit.

Unreliable Connection — A Transport Service Type in which a Queue Pair is associated with only one other QP, such that messages transmitted by the send queue of one QP are, if delivered, delivered to the receive queue of the other QP. As such, each QP is said to be “connected” to the opposite QP. Messages with errors are not retried by the transport, and error handling must be provided by a higher level protocol.

Unreliable Datagram — A Transport Service Type in which a Queue Pair may transmit and receive single-packet messages to/from any other QP. Ordering and delivery are not guaranteed, and delivered packets may be dropped by the receiver.

Unsignaled Completion — A modifier used for Work Requests submitted to the Send Queue signifying that a Work Completion is to be generated only if the requested action completes in error.

Variant CRC — A CRC covering all the fields of a packet, including those that may be changed by Switches.

VCRC — See Variant CRC.

Verbs — An abstract description of the functionality of a Host Channel Adapter. An operating system may expose some or all of the verb functionality through its programming interface.

Verbs Consumer — The direct user of the Verbs.

Virtual Lane — A method of providing independent data streams on the same physical link.

Vital Product Data — Device-specific data to support management functions.

VL — See Virtual Lane.

VPD — See Vital Product Data.

WC — See Work Completion.

Wildcarding — Setting a Component Mask bit in a Management Datagram to 0, causing that component's value to be ignored in carrying out the operation. Not all management classes define a Component Mask and Component Mask is only applicable to certain Method-Attribute combinations. An example of a management class with a Component Mask is SubnAdm.

Window Handle — An opaque object that identifies a Memory Window.

Work Completion — The consumer-visible representation of a Completion Queue Entry. A Work Completion may be obtained when a consumer polls a Completion Queue.

Work Queue — One of Send Queue or Receive Queue.

Work Queue Element — The Host Channel Adapter's internal representation of a Work Request. The consumer does not have direct access to Work Queue Elements.

Work Queue Pair — See Queue Pair.

Work Request — The means by which a consumer requests the creation of a Work Queue Element.

WQ — See Work Queue.

WQE — Work Queue Element, commonly pronounced “wookie”.

WQP — See Work Queue Pair.

WR — See Work Request.

CHAPTER 3: ARCHITECTURAL OVERVIEW

This chapter provides a top-down description of the InfiniBand Architecture (IBA) features, capabilities, components, and elements and it describes various principles of operation. It is a high level overview intended as an informative guide and thus certain details are intentionally excluded for the purpose of clarity.

IBA defines a System Area Network (SAN) for connecting multiple independent processor platforms (i.e., host processor nodes), I/O platforms, and I/O devices (see Figure 6). The IBA SAN is a communications and management infrastructure supporting both I/O and interprocessor communications (IPC) for one or more computer systems. An IBA system can range from a small server with one processor and a few I/O devices to a massively parallel supercomputer installation with hundreds of processors and thousands of I/O devices. Furthermore, the internet protocol (IP) friendly nature of IBA allows bridging to an internet, intranet, or connection to remote computer systems.

IBA defines a switched communications fabric allowing many devices to concurrently communicate with high bandwidth and low latency in a protected, remotely managed environment. An endnode can communicate over multiple IBA ports and can utilize multiple paths through the IBA fabric. The multiplicity of IBA ports and paths through the network are exploited for both fault tolerance and increased data transfer bandwidth.

IBA hardware off-loads from the CPU much of the I/O communications operation. This allows multiple concurrent communications without the traditional overhead associated with communicating protocols. The IBA SAN provides its I/O and IPC clients zero processor-copy data transfers, with no kernel involvement, and uses hardware to provide highly reliable, fault tolerant communications.

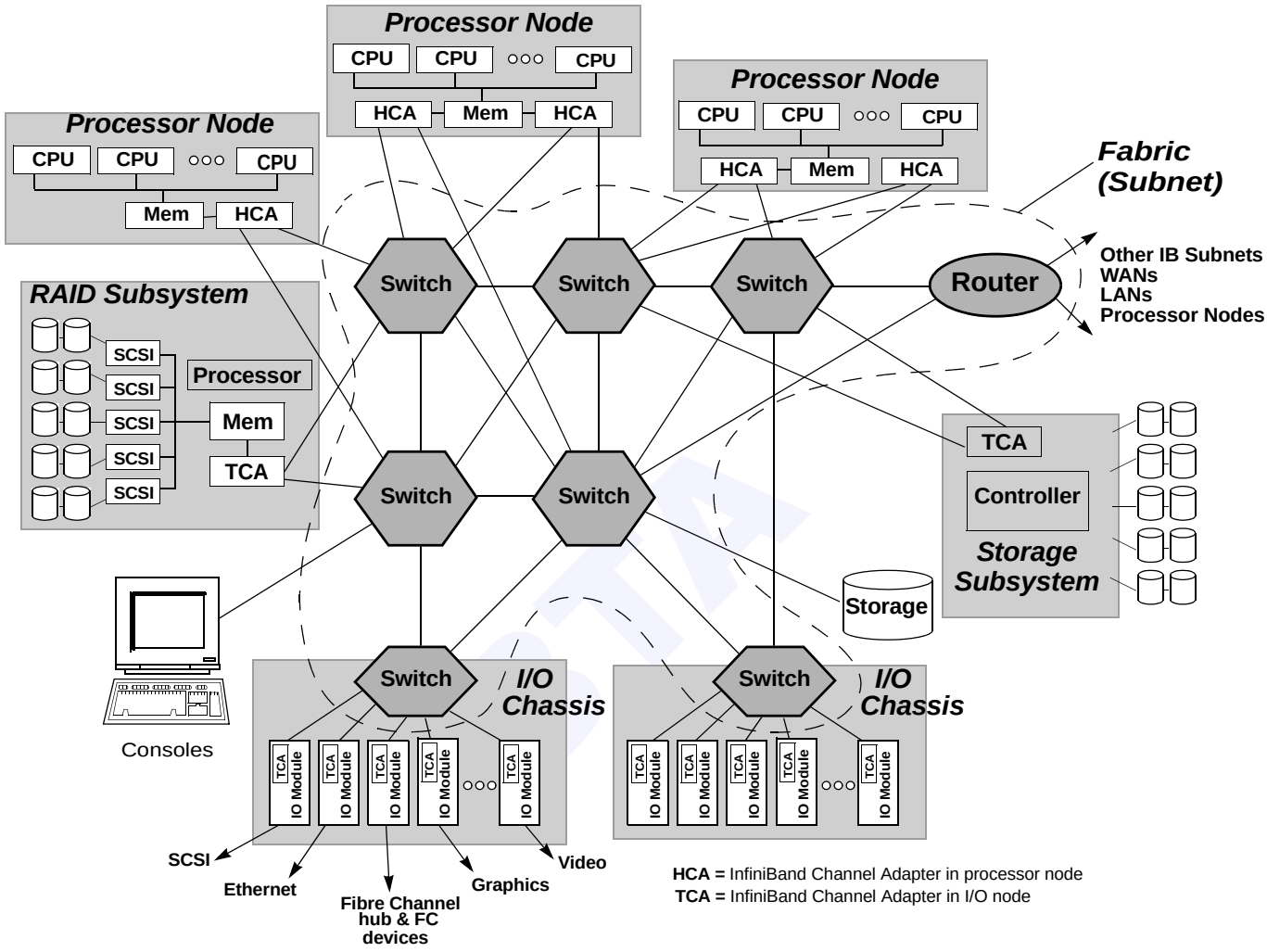


Figure 6 IBA System Area Network

An IBA System Area Network consists of processor nodes and I/O units connected through an IBA fabric made up of cascaded switches and routers.

IO units can range in complexity from single ASIC IBA attached devices such as a SCSI or LAN adapter to large memory rich RAID subsystems that rival a processor node in complexity.

3.1 ARCHITECTURE SCOPE

This volume of the InfiniBand Architecture Specification defines the interconnect fabric, routing elements, endnodes, management infrastructure, and the communication formats and protocols. It does not specify I/O commands or cluster services.

For example, consider an IBA SCSI adapter. IBA does not define the disk I/O commands, how the SCSI adapter communicates with the disk, how the operating system (OS) views the disk device, nor which node in the cluster owns the disk adapter. IBA is an essential underpinning of each of these operations, but does not directly define any of them. Instead, IBA defines how data and commands can be transported between the I/O driver on a processor node and the SCSI adapter.

IBA handles the data communications for I/O and IPC in a multi-computer environment. It supports the high bandwidth and scalability required for IO. It caters to the extremely low latency and low CPU overhead required for IPC. With IBA, the OS can provide its clients with communication mechanisms that bypass the OS kernel and directly access IBA network communication hardware, enabling efficient message passing operation. IBA is well suited to the latest computing models and will be a building block for new forms of I/O and cluster communication. IBA allows I/O units to communicate among themselves and with any or all of the processor nodes in a system. Thus an I/O unit has the same communications capability as any processor node.

3.1.1 TOPOLOGIES & COMPONENTS

At a high level, IBA serves as an interconnect for endnodes as illustrated in Figure 7. Each node can be a processor node, an I/O unit, and/or a router to another network.

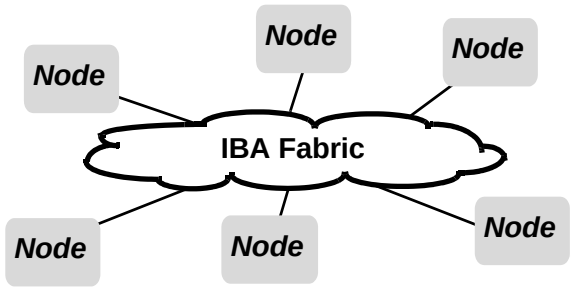


Figure 7 IBA Network

An IBA network is subdivided into subnets interconnected by routers as illustrated in Figure 8. Endnodes may attach to a single subnet or multiple subnets.

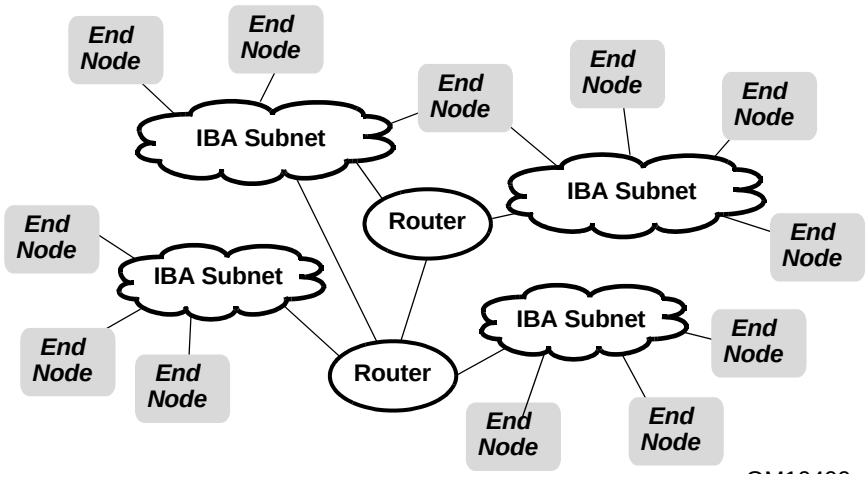


Figure 8 IBA Network Components

An IBA subnet is composed of endnodes, switches, routers, and subnet managers interconnected by links as illustrated in Figure 9. Each IBA device may attach to a single switch or multiple switches and/or directly with each other[1]. Multiple links can exist between any two IBA devices.

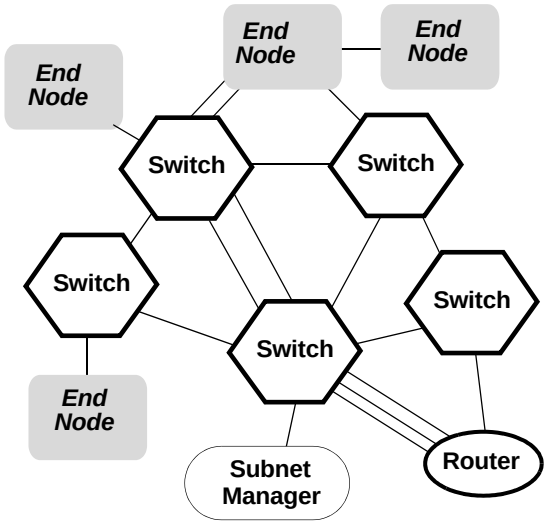


Figure 9 IBA Subnet Components

1. Single endnode to endnode connection creates an independent subnet, with no connectivity to the remainder of the IBA devices, in which case one of the two interconnected endnodes functions as the subnet manager for that link.

The architecture is optimized for units that contain multiple independent processes and threads (consumers) as illustrated in Figure 10. Each channel adapter constitutes a node on the fabric. The architecture supports multiple channel adapters per unit with each channel adapter providing one or more ports that connect to the fabric, in which case the processor node appears as multiple endnodes to the fabric.

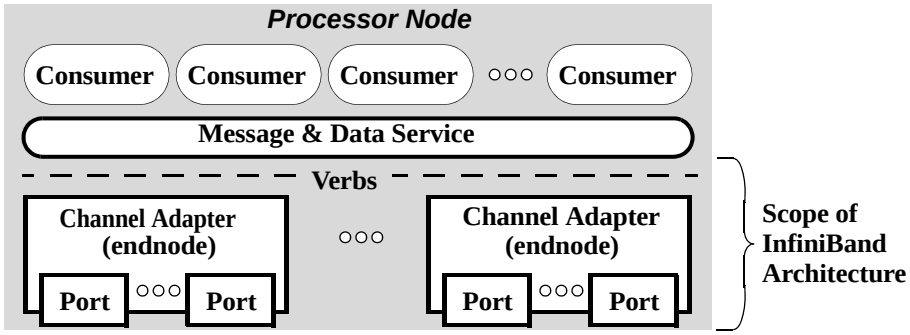


Figure 10 Processor Node

In a processor node, the message and data service is an OS component that is outside the scope of this document. This document specifies the semantic interface between the message and data service and a channel adapter. This semantic interface is referred to as IBA Verbs. Verbs describe the functions necessary to configure, manage, and operate a host channel adapter. These verbs identify the appropriate parameters that need to be included for each particular function. Verbs are not an API, but provide the framework for the OSV to specify the API.

IBA is architected as a first order network and as such it defines the host behavior (verbs) and defines memory operation such that the channel adapter can be located as close to the memory complex as possible. It provides independent direct access between consenting consumers regardless of whether those consumers are I/O drivers and I/O controllers or software processes communicating on a peer to peer basis. IBA provides both channel semantics (send and receive) and direct memory access with a level of protection that prevents access by non participating consumers.

3.2 COMMUNICATION

3.2.1 QUEUING

The foundation of IBA operation is the ability of a consumer to queue up a set of instructions that the hardware executes. This facility is referred to as a work queue. Work queues are always created in pairs, called a Queue Pair (QP), one for send operations and one for receive operations. In general, the send work queue holds instructions that cause data to be transferred between the consumer's memory and another consumer's memory, and the receive work queue holds instructions about where to place data that is received from another consumer. The other consumer

is referred to as a *remote consumer* even though it might be located on the same node. IBA specifically describes the queuing relationship for a *Host Channel Adapter* (HCA) but not the I/O unit because an I/O unit is not necessarily subject to 2nd and 3rd party interoperability that is present in a host environment (i.e., interoperability between the HCA vendor, the OS vendor, and an IHV's I/O driver or an ISV's application using IPC). The following describes the HCA queuing model.

The consumer submits a work request (WR), which causes an instruction called a Work Queue Element (WQE) to be placed on the appropriate work queue. The channel adapter executes WQEs in the order that they were placed on the work queue. When the channel adapter completes a WQE, a Completion Queue Element (CQE) is placed on a completion queue[2]. Each CQE specifies all the information necessary for a work completion, and either contains that information directly or points to other structures, for example, the associated WQE, that contain the information.

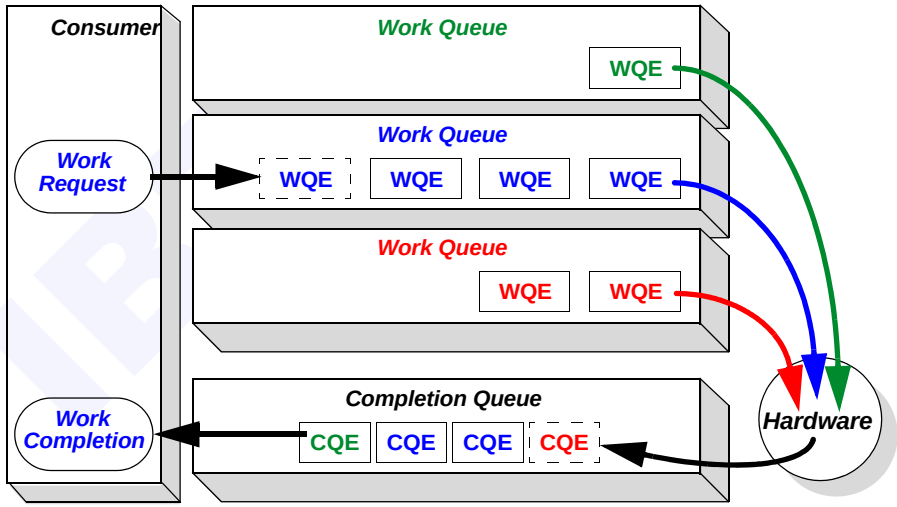


Figure 11 Consumer Queuing Model

Each consumer may have its own set of work queues, each pair of work queues is independent from the others. Each consumer creates one or more completion queues and associates each send and receive queue to a particular completion queue. It is not necessary that both the send and receive queue of a work queue pair use the same completion queue.

Because some work queues require an acknowledgment from the remote node and some WQEs use multiple packets to transfer the data, the channel adapter can have multiple WQEs in progress at the same time, even from the same work queue. Thus the order in which CQEs are

2. WQEs and CQEs are not architected entities, only the Work Request verbs are architected.

posted to the completion queue is not deterministic except that CQEs for the same work queue are normally posted in the order that the corresponding WQE was posted to the work queue[3].

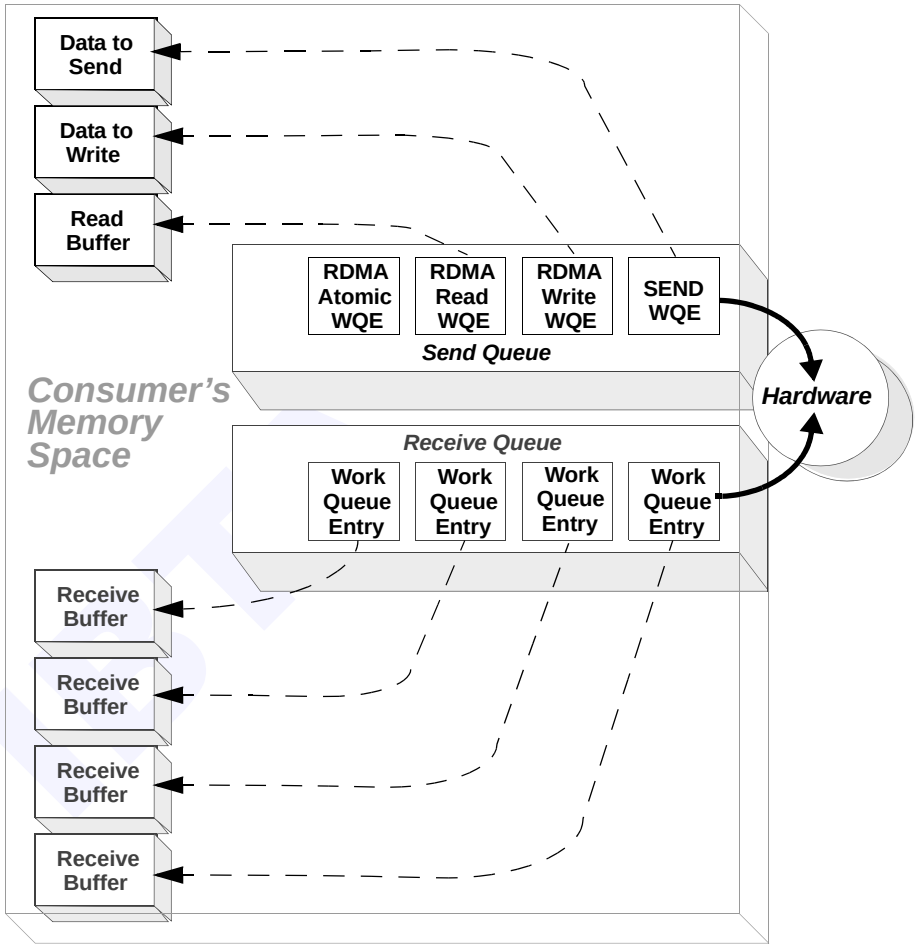


Figure 12 Work Queue Operations

There are three classes of send queue operations SEND, Remote memory Access (RDMA), and MEMORY BINDING.

- For a SEND operation, the WQE specifies a block of data in the consumer's memory space for the hardware to send to the destination, letting a receive WQE already queued at the destination specify where to place that data.

3. Receive completions for reliable datagram service are the exception because concurrent reception on multiple EE contexts can result in out of order posting.

- For an RDMA operation, the WQE also specifies the address in the remote consumer's memory. Thus an RDMA operation does not need to involve the receive work queue of the destination[4]. There are 3 types of RDMA operations, RDMA-WRITE, RDMA-READ, and ATOMIC.
  - The RDMA-WRITE operation stipulates that the hardware is to transfer data from the consumer's memory to the remote consumer's memory.
  - The RDMA-READ operation stipulates that the hardware is to transfer data from the remote memory to the consumer's memory.
  - The ATOMIC operation stipulates that the hardware is to perform a read of a remote 64-bit memory location. The target returns the value read, and conditionally modifies/replaces the remote memory contents by writing an updated value back to the same location.
- MEMORY BINDING instructs the hardware to alter memory registration relationships (see section 10.6.6.2). It associates (binds) a Memory Window to a specified range within an existing Memory Region. Memory binding allows a consumer to specify which portions of registered memory it shares with other nodes (i.e., the memory a remote node can access) and specifies read and write permissions. The result produces a memory key (R_KEY) that the consumer passes to remote nodes for their use in their RDMA operations.

There is only one receive queue operation and it is to specify a receive data buffer.

- A RECEIVE WQE specifies where the hardware is to place data received from another consumer when that consumer executes a SEND operation. Each time the remote consumer successfully executes a SEND operation, the hardware takes the next entry from the receive queue, places the received data in the memory location specified in that receive WQE, and places a CQE on the completion queue indicating to the consumer that the receive operation has completed. Thus the execution of a SEND operation causes a receive queue operation at the remote consumer.

Normally an RDMA operation does not consume a receive WQE at the destination, but there is one exception. That is for an RDMA WRITE operation which specifies immediate data. *Immediate data* is 32 bits of information that is optionally provided in a SEND or RDMA WRITE instruction, transferred as part of the operation, but instead of writing the immediate data to memory, the data is treated as another piece of status information

4. RDMA Write with immediate data does involve the destination's receive work queue.

and returned as a special field of the RECEIVE CQE status. This means that an RDMA WRITE with immediate data will consume a RECEIVE WQE at the destination.

3.2.2 CONNECTIONS

IBA supports both connection oriented and datagram service. For connected service, each QP is associated with exactly one remote consumer. In this case the QP context is configured with the identity of the remote consumer's queue pair. The remote consumer is identified by a port and a QP number. The port is identified by a local ID (LID) and optionally a Global ID (GID). During the communication establishment process, this and other information is exchanged between the two nodes.

For datagram service, a QP is not tied to a single remote consumer, but rather information in the WQE identifies the destination. A communication setup process similar to the connection setup process needs to occur with each destination to exchange that information.

3.3 COMMUNICATIONS STACK

The communication stack for IBA is illustrated in Figure 13. The architecture provides a number of IBA transactions that a consumer can use to execute a transaction with a remote consumer. The consumer posts work queue elements (WQE) to the QP and the channel adapter interprets each WQE to perform the operation.

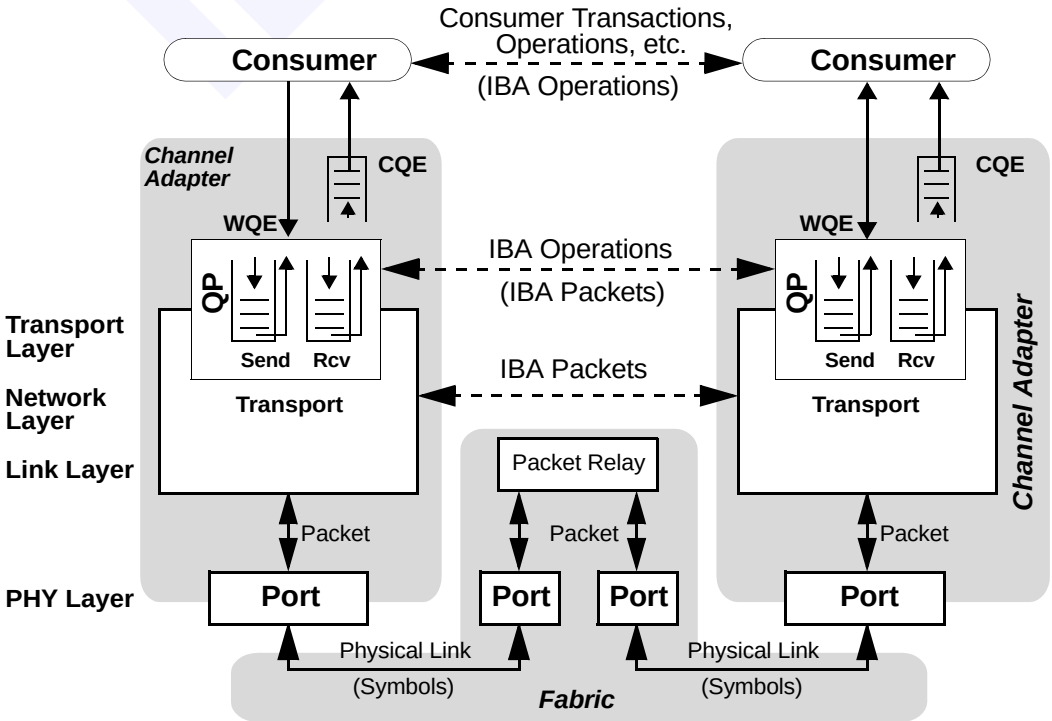


Figure 13 IBA Communication Stack

For Send Queue operations, the channel adapter interprets the WQE, creates a request message. segments the message into multiple packets if necessary, adds the appropriate routing headers, and sends the packet out the appropriate port.

The port logic transmits the packet over the link where switches and routers relay the packet through the fabric to the destination.

When the destination receives a packet, the port logic validates the integrity of the packet. The channel adapter associates the received packet with a particular QP and uses the context of that QP to process the packet and execute the operation. If necessary, the channel adapter creates a response (acknowledgment) message and sends that message back to the originator.

Reception of certain request messages cause the channel adapter to consume a WQE from the receive queue. When it does, a CQE corresponding to the consumed WQE is placed on the appropriate completion queue, which causes a work completion to be issued to the consumer that owns the QP.

3.4 IBA COMPONENTS

The devices in an IBA system are classified as:

- switches
- routers
- channel adapters
- repeaters
- links that interconnect switches, routers, repeaters, and channel adapters

The management infrastructure includes:

- subnet managers
- general service agents

3.4.1 LINKS & REPEATERS

Links interconnect channel adapters, switches, repeaters, and routing devices to form a fabric. A link can be a copper cable, an optical cable, or printed circuit wiring on a backplane. Repeaters are transparent[5] devices that extend the range of a link. Volume 2 of InfiniBand Architecture specifies link and repeater requirements for various media types as well as defines various module form factors for I/O devices. The architecture

5. Transparent in the sense repeaters only participate at the physical layer protocol level and nodes are not aware of their presence.

described in Volume 1 is independent of the type of link and the form factor.

Links and repeaters are not directly addressable but the link status can be determined via the device on each end of the link.

3.4.2 CHANNEL ADAPTERS

Channel adapters are the IBA devices in processor nodes and I/O units that generate and consume packets. IBA defines two types of channel adapters: *Host Channel Adapter* (HCA) and *Target Channel Adapter* (TCA). The HCA provides a consumer interface providing the functions specified by IBA verbs. IBA does not specify the semantics of the consumer interface for a TCA.

A channel adapter is a programmable DMA engine with special protection features that allow DMA operations to be initiated locally and remotely.

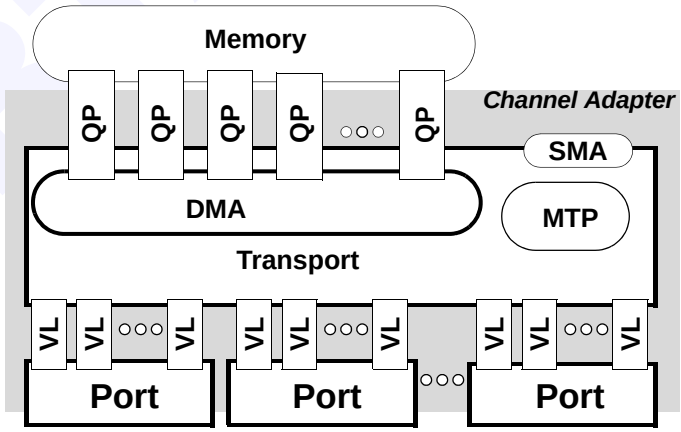


Figure 14 Channel Adapter

A channel adapter may have multiple ports. Each port of a channel adapter is assigned a *Local ID* (LID) or a range of LIDs. Each port has its own set of transmit and receive buffers such that each port is capable of sending and receiving concurrently. Buffering is channeled through *virtual lanes* (VL) where each VL has its own flow control.

The channel adapter provides a Memory Translation & Protection (MTP) mechanism that translates virtual addresses to physical addresses and to validate access rights. Specific memory management mechanisms are not specified by this document, and requirements for such mechanisms are not specified for TCAs.

The channel adapter provides multiple instances of the communication interface to its consumer in the form of *queue pairs* (QP) comprised of a send and receive work queue.

A subnet manager configures channel adapters with the local addresses for each physical port, i.e., the port's LID. The entity that communicates with the subnet manager for the purpose of configuring the channel adapter is referred to as the *Subnet Management Agent* (SMA).

Each channel adapter has a *globally unique identifier* (GUID) assigned by the channel adapter vender. Since local IDs assigned by the subnet manager are not persistent (i.e., might change from one power cycle to the next), the channel adapter GUID (called Node GUID) becomes the primary object to use for persistent identification of a channel adapter. Additionally, each port has a Port GUID assigned by the channel adapter vender.

3.4.3 SWITCHES

Switches primarily pass packets along based on the destination address in the packet's local route header.a switch also consumes and sources packets required for managing the switch itself. Optionally, a switch port may incorporate the properties of a physical TCA port.

IBA switches are the fundamental routing component for intra-subnet routing (inter-subnet routing is provided by IBA routers). Switches interconnect links by relaying packets between the links.

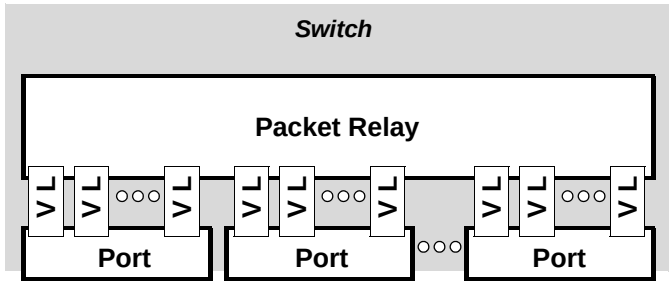


Figure 15 IBA Switch Elements

Switches expose two or more ports between which packets are relayed.

Every destination within the subnet is configured with one or more unique local identifiers (LIDs). Packets contain a destination address that specifies the LID of the destination. From the point of view of a switch, the Destination LID represents a path through the switch. Switch elements are configured with forwarding tables. Individual packets are forwarded within a switch to an outbound port or ports based on the packet's Destination LID and the Switch's forwarding table.

IBA switches support unicast forwarding and may support multicast forwarding. Unicast is the delivery of a single packet to a single destination and multicast is the ability of the fabric to deliver a single packet to multiple destinations.

A subnet manager configures switches including loading their forwarding tables.

To maximize availability, multiple paths between endnodes may be deployed within the switch fabric. If multiple paths are available between switches, the subnet manager can use these paths for redundancy or for destination LID based load sharing. Where multiple paths exists, a subnet manager can re-route packets around failed links by re-loading the forwarding tables of switches in the affected area of the fabric.

3.4.4 ROUTERS

Like switches, routers do not generate nor consume packets (except for management packets). They simply pass them along. Routers forward packets based on the packet's global route header and actually replaces the packet's local route header as the packet passes from subnet to subnet.

IBA routers are the fundamental routing component for inter-subnet routing (intra-subnet routing is provided by IBA switches). Routers interconnect subnets by relaying packets between the subnets.

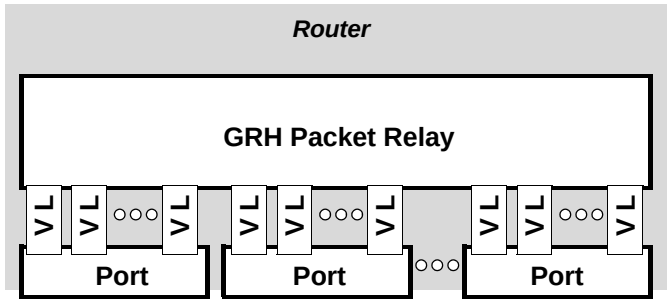


Figure 16 IBA Router Elements

Routers expose one or more ports between which packets are relayed. Routers could be embedded with other devices, such as channel adapters or switches.

Routers are not completely transparent to the endnodes since the source must specify the LID of the router and also provide the GID of the destination.

Each subnet is uniquely identified with a subnet ID known as the Subnet Prefix. The subnet manager programs all ports (via the PortInfo attribute) with the Subnet Prefix for that subnet. When combined with a Port GUID, this combination becomes a port's natural GID. Ports may have other locally administrated GIDs.

From the point of view of a router, the subnet prefix portion of a GID represents a path through the router. IPv6 specifies the protocol performed between routers to derive their forwarding tables. Individual packets are forwarded within a router to an outbound port or ports based on the packet's Destination GID and the router's forwarding table.

Each router forwards the packet through the next subnet to another router until the packet reaches the target subnet. The last router sends the packet using the LID associated with the Destination GID as the Destination LID.

A subnet manager configures routers with information about the subnet such as which VLs to use and partition information.

To maximize availability, multiple paths between subnets may be deployed within the fabric. If multiple paths are available, routers might use those paths for redundancy or for load sharing. Where multiple paths exist, a router can re-route packets around failed subnets.

3.4.5 MANAGEMENT COMPONENTS

IBA management provides for a subnet manager and an infrastructure that supports a number of general management services. The management infrastructure requires a subnet management agent in each node and defines a general service interface that allows additional general services agents.

The architecture defines a common management datagram (MAD) message structure for communicating between managers and management agents.

3.4.5.1 SUBNET MANAGERS

A *Subnet Manager* (SM) is an entity attached to a subnet that is responsible for configuring and managing switches, routers, and channel adapters. A SM can be implemented with other devices, such as a channel adapter or a switch.

IBA supports the notion of multiple subnet managers per subnet and specifies how multiple subnet managers negotiate for one to become the master SM. It does not prohibit other methods between cooperating SMs for governing master/standby relationships

The master SM:

- discovers the subnet topology,
- configures each channel adapter port with a range of LIDs, GIDs subnet prefix, and P_Keys,
- configures each switch with a LID, the subnet prefix, and with its forwarding database,
- maintains the endnode and service databases for the subnet and thus provides a GUID to LID/GID resolution service as well as a services directory.

3.4.5.2 SUBNET MANAGEMENT AGENTS

Each node provides a Subnet Management Agent (SMA) that the SM access through a well known interface called the Subnet Management Interface (SMI). SMI allows for both LID Routed packets and Directed Routed packets. Directed routing provides the means to communicate before switches and end nodes are configured. Only the SMI allows for directed routed packets.

3.4.5.3 GENERAL SERVICE AGENTS

Each node may contain additional management agents referred to as General Service Agents (GSA*) that can be accessed through a well known interface called the General Service Interface (GSI). The GSI only supports LID routing. The general service classes defined by IBA are:

- Subnet Administration (SA) - this is a service provided by the SM that allows nodes to access information about the subnet to discover other nodes and services, to resolve paths, and to register its services.
- Performance Management - monitors and reports well-defined performance counters
- Baseboard Management - provides for chassis management using IB-ML as defined in Volume 2.
- SNMP Tunneling - provides SNMP functionality by defining the method for sending and receiving SNMP messages.
- Vendor Defined - allows private extensions that a device vendor may use to remotely configure and manage its devices.
- Communication Management (ComMgt) - Provides for connection establishment and other communication management functions between endnodes.
- Device Management (DevMgt) - Provides I/O resource management

3.5 IBA FEATURES

3.5.1 QUEUE PAIRS

The QP is the virtual interface that the hardware provides to an IBA consumer and it provides a virtual communication port for the consumer. The architecture supports up to 2^{24} QPs per channel adapter and the operation on each QP is independent from the others. Each QP provides a high degree of isolation and protection from other QP operations and other consumers. Thus a QP can be considered a private resource assigned to a single consumer. A consumer might consume multiple QPs as illustrated in Figure 17.

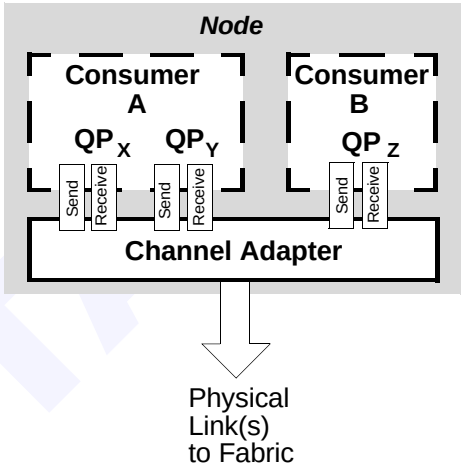


Figure 17 Communication Interface

The consumer creates this virtual communication port by allocating a QP and specifying its class of service. Communication takes place between a source QP and a destination QP. For connection oriented service, each QP is tightly bound to exactly one other QP, usually on a different node. The consumer initiates any communication establishment necessary to bind the QP with the destination QP and configures the QP context with certain information such as Destination LID, service level, and negotiated operating limits.

The consumer posts work requests to a QP to invoke communication through that QP.

3.5.2 TYPES OF SERVICE

Each QP is configured for a certain class of operation (referred to as service type) based on how the sourcing and receiving QPs interact. Both the source and destination QPs must be configured for the same service type. Each service type is based on the following attributes.

- **Connection oriented** versus **datagram** - For connection oriented service, the QP is associated with exactly one other QP and all work requests posted to the QP results in a message sent to the established destination QP. Datagram operation allows a single QP to be used to send and receive messages to/from any appropriate QP on any node.
- **Acknowledged** versus **unacknowledged** - For acknowledged service, a QP returns response messages when it receives request messages. Response messages might be positive acknowledgment (ACK), negative acknowledgment (NAK), or contain response data. Acknowledged service is referred to as *reliable* since the transport protocol guarantees un-corrupted data delivery, in order, exactly once. Unacknowledged service is referred to as unreliable because the transport protocol does not guarantee that all data is delivered. It does guarantee that all data is delivered at most once, and delivered data is not corrupted. Also there are certain cases where changes in fabric configuration might cause data to be delivered out of order.
- **IBA transport** versus **other transport** - IBA transport services define a specific transport protocol for channel based and memory based operations. IBA also supports using the channel adapter as a data link engine to send raw packets between nodes which is useful for supporting legacy protocol stacks and legacy networks.

The service types defined by IBA are specified in Table 2

Table 2 Service Types

Service Type	Connection Oriented	Acknowledged	Transport
Reliable Connection	yes	Yes	IBA
Unreliable Connection	yes	no	IBA
Reliable Datagram	no	Yes	IBA
Unreliable Datagram	no	no	IBA
RAW Datagram	no	no	Raw

Certain IBA operations are valid only over certain classes of service. A QP rejects a WQE for an operation that is not valid for the configured class of service.

Connection oriented service requires that the consumer initiate a communication establishment procedure (connection setup) with the target node to associate the QPs and establish QP context prior to any QP operation. Actually, all service classes except for raw datagram need some form of

communication setup to associate queue pairs. For reliable datagram service, the node performs a communication establishment process to associate an end-to end (EE) context (explained later) with each target node. All QPs configured for Reliable Datagram service use established EE contexts and the work request specifies which EE context to use for that operation.

Raw Datagrams are similar to unreliable datagrams, except that the source QP does not know the identity of the QP that will receive and process the message. Raw datagrams allow for routers that forward raw datagram packets to non IBA destinations on a disparate fabric (such as a LAN or WAN) that has no equivalent of a QP. There are two types of raw datagrams, IPv6 and Ethertype. IPv6 raw datagrams contain a global routing header and the packet payload contains a transport protocol service data unit as identified in the global routing header. An Ethertype raw datagram contains an Ethernet Type field and the packet payload contains a transport protocol service data unit as identified in the Ethernet Type field.

IBA defines both channel (send/receive) and memory (RDMA) semantics. Raw datagram and Unreliable Datagram services do not support memory semantics.

3.5.3 KEYS

IBA uses various keys to provide isolation and protection. Keys are values assigned by an administrative entity that are used in messages in various ways. The keys themselves do not provide security since the keys are available in messages that cross the fabric and thus any entity that can get to the interior of the fabric can ascertain key values. IBA does place restrictions on how applications can access certain keys.

The keys are:

- **Management Key** (M_Key): Enforces the control of a master subnet manager. Administered by the subnet manager and used in certain subnet management packets. Each channel adapter port has a M_Key that the SM sets and then enables. The SM may assign a different key to each port. Once enabled, the port rejects certain management packets that do not contain the programmed M_Key. Thus only a SM with the programed M_Key can alter a node's fabric configuration. The SM can prevent the port's M_Key from being read as long as the SM is active. The port maintains a time-out such that the port reverts to an unmanaged state if the SM fails. There is one M_Key for a switch.
- **Baseboard Management Key** (B_Key): Enforces the control of a subnet baseboard manager. Administered by the subnet baseboard manager and used in certain MADs. Each channel adapter port has a

B_Key that the baseboard manager sets. The baseboard manager may assign a different key to each port. Once enabled, the port rejects certain management packets that do not contain the programmed B_Key. Thus only a baseboard manager with the programed B_Key can alter a node's baseboard configuration. The baseboard manager can prevent the port's B_Key from being read as long as the baseboard manager is active. The port maintains a timeout such that the port reverts to an unmanaged state if the baseboard manager fails. There is one B_Key for a switch.

- **Partition Key** (P_Key): Enforces membership. Administered through the subnet manager by the partition manager (PM). Each channel adapter port contains a table of partition keys which is setup by the PM. QPs are required to be configured for the same partition to communicate (except QP0, QP1, and ports configured for raw datagrams) and thus the P_Key is carried in every IB transport packet. Part of the communication establishment process determines which P_Key that a particular QP or EEC uses. An EEC contains the P_Key for Reliable Datagram service and a QP context contains the P_Key for the other IBA transport types. The P_Key in the QP or EEC is placed in each packet sent, and compared with the P_Key in each packet received. Received packets whose P_Key comparison fails are rejected. Each switch has one P_Key table for management messages and may optionally support partition enforcement tables that filter packets based on their P_Key.

- **Queue Key** (Q_Key): Enforces access rights for reliable and unreliable datagram service (RAW datagram service type not included). Administered by the channel adapter. During communication establishment for datagram service, nodes exchange Q_Keys for particular queue pairs and a node uses the value it was passed for a remote QP in all packets it sends to that remote QP. Likewise, the remote node uses the Q_Key it was provided. Receipt of a packet with a different Q_Key than the one the node provided to the remote queue pair means that packet is not valid and thus rejected.

  Q_Keys with the most significant bit set are considered controlled Q_Keys (such as the GSI Q_Key) and a HCA does not allow a consumer to arbitrarily specify a controlled Q_Key. An attempt to send a controlled Q_Key results in using the Q_Key in the QP context. Thus the OS maintains control since it can configure the QP context for the controlled Q_Key for privileged consumers.

- **Memory Keys** (L_Key and R_Key): Enables the use of virtual addresses and provides the consumer with a mechanism to control access to its memory. These keys are administered by the channel adapter through a registration process. The consumer registers a region of memory with the channel adapter and receives an L_Key and R_Key. The consumer uses the L_Key in work requests to describe local memory to the QP and passes the R_Key to a remote consumer

for use in RDMA operations. When a consumer queues up a RDMA operation it specifies the R_Key passed to it from the remote consumer and the R_Key is included in the RDMA request packet to the original channel adapter. The R_Key validates the sender's right to access the destination's memory and provides the destination channel adapter with the means to translate the virtual address to a physical address.

3.5.4 VIRTUAL MEMORY ADDRESSES

IBA is optimized for virtual addressing. That is, an IBA consumer uses virtual addresses in work requests and the channel adapter is able to convert the virtual address to physical address as necessary. For this to happen, each consumer registers regions of virtual memory with the channel adapter and the channel adapter returns 2 memory handles called L_Key and R_Key to the consumer. The consumer then uses the L_key in each work request that requires a memory access to that region. See Section 3.5.3 for description of L_Key usage.

Memory Registration provides mechanisms that allow IBA consumers to de-scribe a set of virtually contiguous memory locations or a set of physically contiguous memory locations to allow the HCA to access the memory as a virtually contiguous buffer using virtual addresses.

IBA also supports remote memory access (RDMA) that permits a remote consumer to access that registered memory. For RDMA, the consumer passes the R_KEY and a virtual address of a buffer in that memory region to another consumer. That remote consumer supplies that R_Key in its RDMA WQEs that will access memory in the original node. See Section 3.5.3 for detailed description of R_Key usage.

3.5.5 PROTECTION DOMAINS

Not only does memory registration allow the use of virtual memory addressing, but it also provides an increased level of protection against inadvertent and unauthorized access.

Since a consumer might communicate with many different destinations but not wish to let all those destinations have the same access to its registered memory, IBA provides protection domains. Protection domains allow a consumer to control which set of its Memory Regions and Memory Windows can be accessed by which set of its QPs.

Before a consumer allocates a QP or registers memory, it creates one or more protection domains. QPs are allocated to, and memory registered to, a protection domain. L_Keys and R_Keys for a particular memory domain are only valid on QPs created for the same protection domain.

3.5.6 PARTITIONS

Partitioning enforces isolation among systems sharing an InfiniBand fabric. Partitioning is not related to boundaries established by subnets, switches, or routers. Rather a partition describes a set of endnodes within the fabric that may communicate.

Each port of an endnode is a member of at least one partition and may be a member of multiple partitions. A partition manager assigns partition keys (P_Keys) to each channel adapter port. Each P_Key represents a partition. Each QP[6] and EE context is assigned to a partition and uses that P_Key in all packets it sends and inspects the P_Key in all packets it receives. Reception of an Invalid P_Key causes the packet to be discarded.

Switches and routers may optionally be used to enforce partitioning. In this case the partition manager programs the switch or router with P_Key information and when the switch or router detects a packet with an invalid P_Key, it discards the packet.

3.5.7 VIRTUAL LANES

Virtual lanes (VL) provide a mechanism for creating multiple virtual links within a single physical link. A virtual lane represents a set of transmit and receive buffers in a port. All ports support VL_{15} which is reserved exclusively for subnet management. There are 15 other VLs (VL_0 to VL_{14}) called data VLs and all ports support at least one data VL (VL_0) and may provide VL_1 to VL_{n-1}, where n is the number of data VLs the port supports).

The actual data VLs that a port uses is configured by the SM and is based on the Service Level (SL) field in the packet. The default is to use VL_0 until the SM determines the number of VLs that are supported by both ends of the link and programs the port's SL to VL mapping table.

6. Except QP0, QP1, and QPs configured for Raw Datagrams type of service.

The port maintains separate flow control over each data VL such that excessive traffic on one VL does not block traffic on another VL.

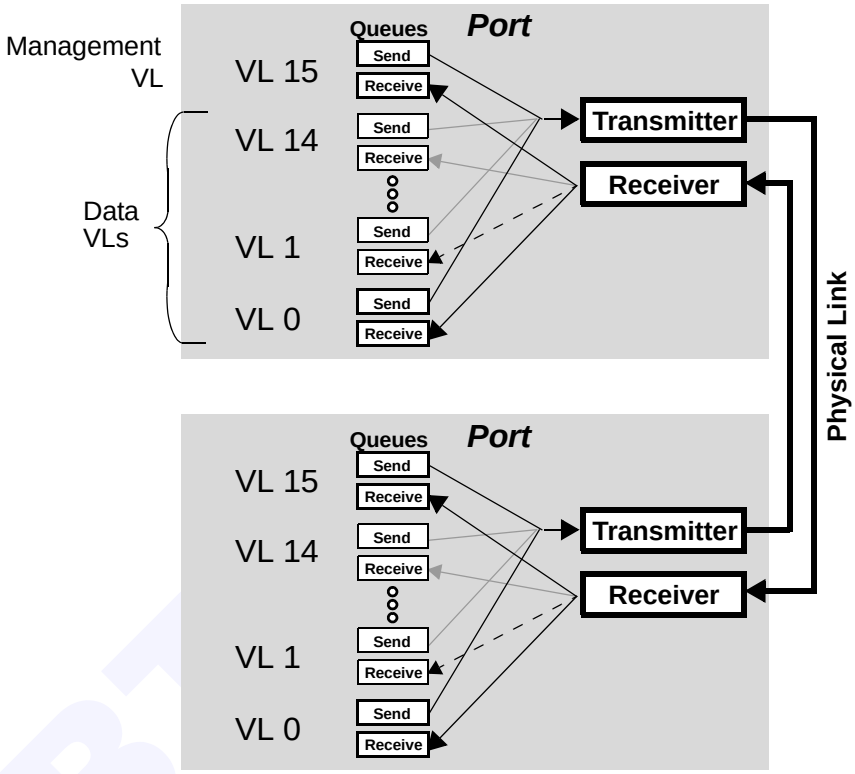


Figure 18 Virtual Lanes

VL assignment exists only between ports at each end of a link and VL assignment on one link is independent of assignments on other links.

Each packet has a SL which is specified in the packet header. As a packet traverses the fabric, its SL determines which VL will be used on each link. Each port maintains a table of SL to VL mapping such that a packet is sent on the appropriate VL.

When the ports at each end of a link support a different number of data VLs, the port with the higher number degrades to the number supported by the other port. Thus for ports that only support a single data VL, all data traffic defaults to VL_0.

3.5.8 QUALITY OF SERVICE

IBA provides several mechanisms that permit a subnet manager to administer various quality of service guarantees for both connected and connectionless services. These mechanisms are Service Level, Service Level to Virtual Lane Mapping, and Partitions. IBA does not define quality of service (QoS) levels (e.g., best effort).

3.5.8.1 SERVICE LEVEL

IBA defines a Service Level (SL) attribute that permits a packet to operate at one of 16 service levels. The definition and purpose of each service level is outside the scope of the architecture and left as a fabric administration policy. Thus the assignment of service levels is a function of each node's communication manager and its negotiation with a subnet manager.

3.5.8.2 SL TO VL MAPPING

Another IBA mechanisms that is tied to service levels is virtual lanes. Each packet identifies its SL and as the packet traverses the fabric, the packet's SL determines which VL is used on the next link. To this end, each port (switches, routers, endnodes) has a SL to VL mapping table that is configured by subnet management. Naturally, for all links that terminate at a port that only supports one data VL, all SLs map to VL_0. Otherwise, subnet management policy determines the mapping of each SL to an available VL.

Packets addressed to QP0 are Subnet Management Packets (SMP) and exclusively use VL15 and their SL is ignored. VL15 (the management VL) is not a data VL and is not used for packets not addressed to QP0.

3.5.8.3 PARTITIONS

Another IBA mechanism that can be tied to service levels is partitioning. Fabric administration can assign certain SLs for particular partitions. This allows the SM to isolate traffic flows between those partitions and even if both partitions operate at the same QoS level, each partition can be guaranteed its fair share of bandwidth regardless of whether nodes in other partitions misbehave or are over subscribed.

3.5.9 INJECTION RATE CONTROL

IBA defines a number of different link bit rates. The lowest bit rate of 2.5 Gb/sec is referred to as a 1x (times one) link. Other link rates are 10Gb/sec (4x) and 30 Gb/sec (1x2). To support multiple link speeds within a fabric, IBA defines a *Static Rate Control* mechanism that prevents a port with a high speed link from overrunning the capacity of a port with a lower speed link.

As part of the path resolution process, the SubnAdm:PathRecord provides the node with MTU and rate information for the path. Path information is used since either a switch port or the endnode could be the limiting factor.

The example in Figure 19 illustrates that port A with a 12x link speed has the potential for injecting traffic at 3 times the capacity of port B and 12 times the capacity of ports C, D, or E. Additionally port B has the potential

for injecting traffic at 4 times the capacity of port C, D, or E. Since traffic tends to be bursty, every time port A sends to one of the other ports, the fabric has a high probability of congesting. Link flow control keeps the fabric from losing packets due to that congestion, but the back pressure will effect other paths that otherwise would not be congested.

IBA solves this problem by defining a static rate control mechanism for ports that operate at link speeds greater than 1x.

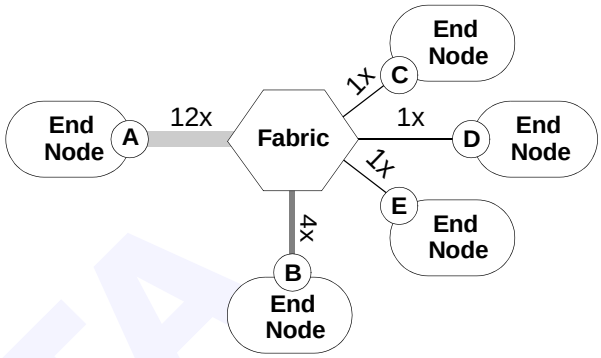


Figure 19 Rate Matching Example

Each destination has a time-out value associated with it and that time-out value is based on the ratio between the source and destination bit rates. When the source and destination bit rates are equal, the time-out values is 0 (not needed). Otherwise when the port transmits a packet to a destination, it puts that destination LID and a time-out value in its static rate control table. The port removes the entry after the time-out period expires. While the entry remains in the table, the port does not send any more packets to that destination (i.e., defers to traffic for other destinations not in the table). When there is no entry in the table, the port transmits the packet by placing it on the appropriate VL output queue.

3.5.10 ADDRESSING

Each endnode contains one or more channel adapters and each channel adapter contains one or more ports. Additionally each channel adapter contains a number of queue pairs (QP).

Each QP has a queue pair number (QPN) assigned by the channel adapter which uniquely identifies the QP within the channel adapter. There are two well-known QPs for each port (QP0 and QP1) and all other QPs are configured for operation through a particular port. For reliable datagram service, it is the EE context rather than the QP context that determines the port.

Packets other than raw datagrams contain the QPN of the destination QP. When the channel adapter receives a packet, it uses the context of the destination QPN (and EE context for reliable datagram) to process the packet.

Each port has a Local ID (LID) assigned by the local subnet manager (i.e., the subnet manager for the subnet). Within the subnet, LIDs are unique. Switches use the LID to route packets within the subnet. The local subnet manager configures routing tables in switches based on LIDs and where that port is located with respect to the specific switch. Each packet contains a Source LID (SLID) that identifies the port that injected the packet into the subnet and a Destination LID (DLID) that identifies the port where the fabric is to deliver the packet.

IBA also provides for multiple virtual ports within a physical port by defining a LID Mask Control (LMC). The LMC specifies the number of least significant bits of the LID that a physical port masks (ignores) when validating that a packet DLID matches its assigned LID. Those bits are not ignored by switches, therefore the subnet manager can program different paths through the fabric based on those least significant bits. Thus the port appears to be 2^{LMC} ports for the purpose of routing across the fabric.

Each port also has at least one Global ID (GID) that is in the format of an IPv6 address. GIDs are globally unique. Each packet optionally contains a Global Route Header (GRH) specifying a Source GID (SGID) that identifies the port that injected the packet into the fabric and a Destination GID (DGID) that identifies the port where the fabric is to deliver the packet. Routers use the GRH to route packets between subnets. Switches ignore the GRH.

Each channel adapter has a Globally Unique Identifier (GUID) called the Node GUID assigned by the channel adapter vendor. Each of its ports has a Port GUID also assigned by the channel adapter vendor. The Port GUID combined with the local subnet prefix becomes a port's default GID.

Subnet administration provides a GUID to LID/GID resolution service. Thus a node can persistently identify another node by remembering a Node or Port GUID.

The address of a QP is the combination of the port address (GID + LID) and the QPN. To communicate with a QP requires a vector of information including the port address (LID and/or GID), QPN, service level, path MTU, and possibly other information. This information can be obtained by a path query request addressed to Subnet Administration.

Service IDs are used to resolve QPs. Some Service IDs are well known (i.e., certain functions have a predetermined Service ID) and some are advertised in an I/O controller's Service Entries list. The subnet manager

provides the GUID to GID/LID resolution, but the target provides a Service ID to QP resolution as part of the communication management process.

In general, the target node of a Request for Communication message uses the Service ID to direct the request to the entity who decides whether to proceed with communication establishment. If the decision is affirmative, the target returns the information necessary to establish communication, which includes the QPN plus other information specific to the transport service type.

A simplified address resolution process is illustrated in Figure 20.

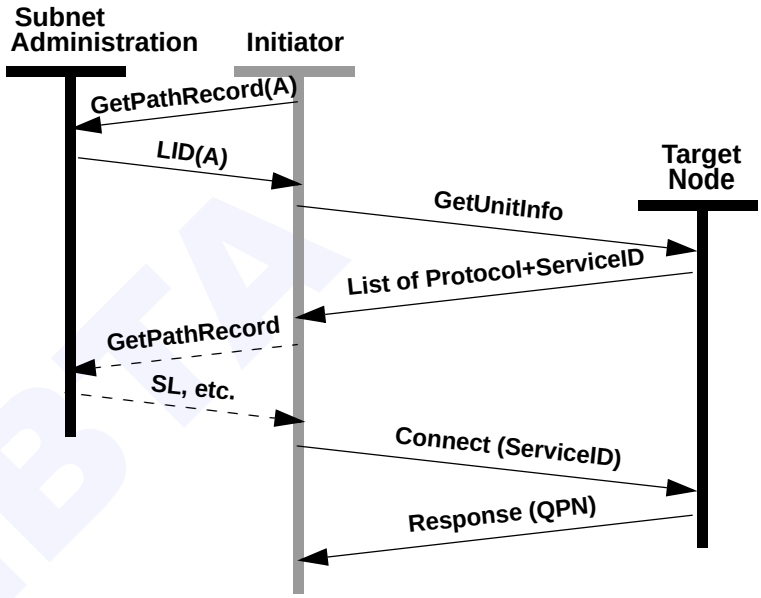


Figure 20 Simplified Address Resolution Process

In the illustration, the target is an I/O controller where the initiator learns the Service ID by querying the IOC for a list of I/O protocols supported. The second path resolution is only necessary if the service being established uses different path characteristics (SL, QoS, MTU, etc.) than the management MADs.

3.5.11 MULTICAST

Multicast is a one-to-many / many-to-many communication paradigm designed to simplify and improve the efficiency of communication between a set of endnodes.

Each multicast group is identified by a unique GID. A node joins and leaves a multicast group through a management action where the node supplies the LID for each port that will participate. This information is distributed to the switches. Each switch is configured with routing information for the multicast traffic which specifies all of the ports where the packet needs to travel. Care is taken to assure there are no loops (i.e., a single

spanning tree such that a packet is not forwarded to a switch that already processed that packet).

The node uses the multicast LID and GID in all packets it sends to that multicast group. When a switch receives a multicast packet (i.e., a packet with a multicast LID in the packet's DLID field) it replicates the packet and sends it out to each of the designated ports except the arrival port. In this fashion, each cascaded switch replicates the packet such that the packet arrives only once at every subscribed endnode.

The channel adapter may limit the number of QPs that can register for the same multicast address. The channel adapter distributes multicast packets to QPs registered for that multicast address. A single QP can be registered for multiple addresses for the same port but if a consumer wishes to receive multicast traffic on multiple ports it needs a different QP for each port. The channel adapter recognizes a multicast packet by the packet's DLID or by the special value in the packet's Destination QP field and routes the packet to the QPs registered for that address and port. Note that the Destination QP field in a multicast packet is not a QPN.

3.5.11.1 MULTICAST EXAMPLE

Figure 21 illustrates an example unreliable multicast IBA operation:

- A packet with PSN = 1129 is received on an IBA routing element (switch or router) port.
- The switching / routing element examines the packet header and extracts the DLID / multicast GID to determine if it corresponds to a multicast group. An implementation may maintain this data as part of its internal route table, e.g. a bit-mask which corresponds to the output ports this packet should be forwarded.
- Switches or routers replicate the packet (implementation dependent) and forwards the packet onto the output port(s)

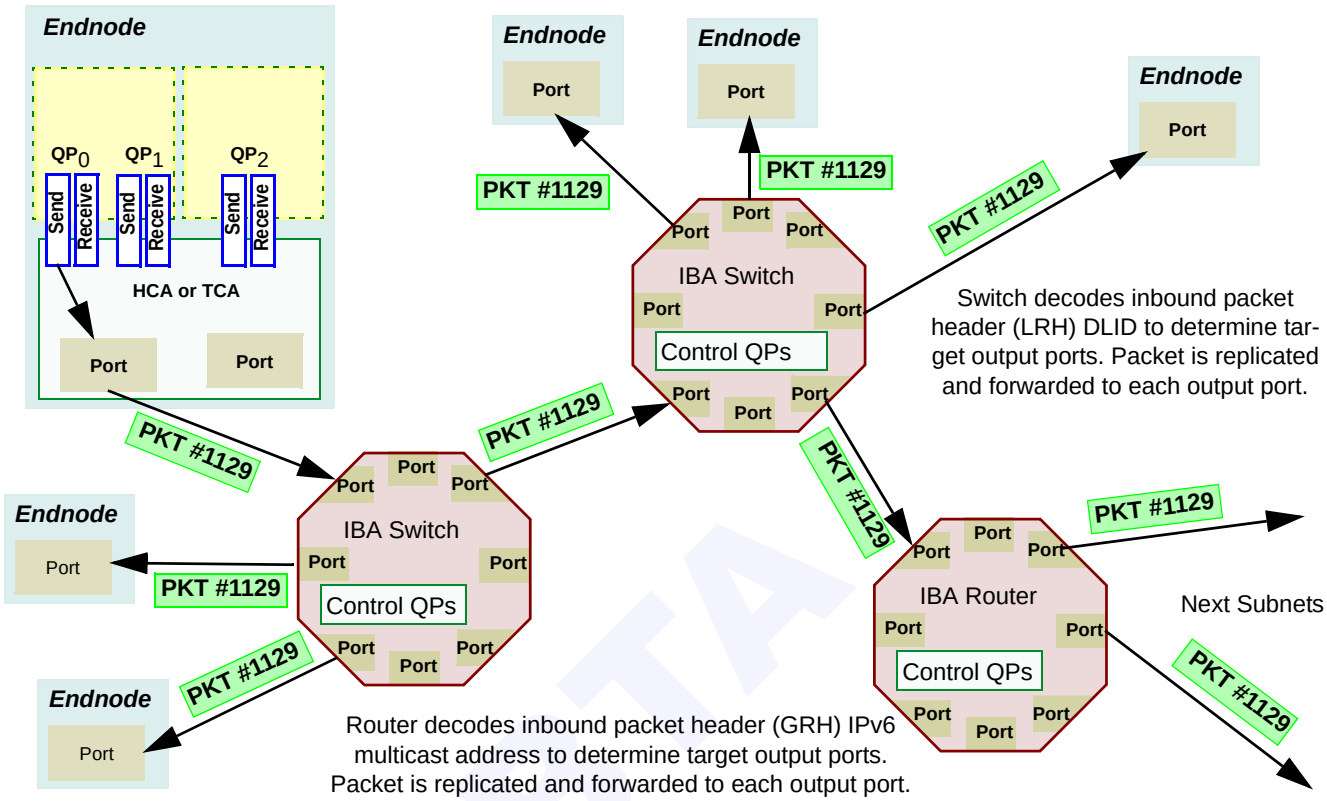


Figure 21 Example Unreliable Multicast Operation

3.5.11.2 GROUP MANAGEMENT

IBA does not define the multicast group management protocol to implement join and leave operations. However, the management interface and associated MADs to implement a multicast group protocol is specified. While these mechanisms are part of the Subnet Administration (SA), some actions are implicitly performed by the Subnet Manager (SM). For the following discussion, the term multicast management entity is used to describe the SA/SM expected responsibility with respect to multicast management. Refer to the Subnet Administration attributes of Multicast Member Record for more information.

3.5.11.2.1 MULTICAST GROUP CREATE

The multicast group creation is an explicit operation in IBA, in order to provide a single control of group characteristics and allow members to join subversively. The group has to be created by the multicast management entity before a join can be successful:

1) An (administrative) application defines (or determines) a target multicast group address (GID). It specifies particular group characteristics (PMTU, P-Key, etc.) and creates the multicast group by invoking a

multicast group create to the multicast management entity. This application may request a specific multicast GID or have one allocated for it.

2) The multicast management entity may notify appropriate routers on the subnet of the new group which is being created (not defined in IBA V 1.1). The router protocol should determine whether this multicast group is in operation within another subnet. If so the router returns the PMTU of the existing multicast group to determine whether the create is allowed or not.

3) The multicast management entity maps the multicast group address to a multicast LID.

3.5.11.2.2 MULTICAST GROUP JOIN

The multicast group join algorithm (applies to IBA unreliable datagram multicast groups) is defined as follows:

1) Application defines or determines the target multicast group address and invokes a multicast join operation.

2) The underlying join implementation determines if the associated endnode is participating in the multicast group. If it is, the application establishes a new local QP and performs the steps required to join this group. If not, the application invokes the management interface to communicate with the multicast group management entity.

3) The multicast management entity performs the following steps upon receiving a join request:

   a) Validate the multicast group address - fail join operation if invalid.

   b) Validate the requested PMTU - fail join operation if invalid.

   c) Verify the switch attached to the endnode is capable of multicast operation. The switch either supports multicast operation via packet replication or it can be configured to send all multicast packets to the endnode-attached port.

   d) If the multicast group address is currently in operation within this subnet, take the following actions:

      i) Verify all switches and routers which are participating in this multicast group can support the requested PMTU. If they cannot, the join operation fails.

      ii) Each multicast group is implemented by defining a logical routing tree across the participating switches. Rebuild / modify the routing tree to include the new endnode. The multicast management entity informs fabric management to update the associated route forwarding tables within all switches and routers to reflect this new topology.

e) If the multicast group address is not operating within this subnet, take the following steps.

i) Inform each router within this subnet of the join operation. The router protocol should determine whether this multicast group is in operation within another subnet. If so, the router returns the PMTU of the existing multicast group to determine whether the create and subsequent join operation is allowed or not.

ii) Map the multicast group address to an unused multicast LID.

iii) Establish a multicast routing tree and update the associated switch and router route forwarding tables accordingly.

iv) Create the group and assign the PMTU to the multicast group.

v) Return the multicast LID and associated group characteristics to the endnode and allow multicast operations to be initiated.

f) Each router within this subnet is informed of successful multicast join operation. Routers invoke the appropriate multicast group management operations to add this subnet as participating in the associated multicast group. This protocol is outside the IBA specification.

4) Add the member to the group.

3.5.11.2.3 MULTICAST GROUP LEAVE

When an application leaves a multicast group, the following algorithm is used:

1) The application's QP is removed as a target for the multicast group. If there are QPs still participating in this multicast group, no further action is required.

2) If there are no more QPs on this port participating within the multicast group, the leave implementation informs the multicast management entity that this endnode is no longer participating in this multicast group. The multicast management entity takes the following step:

a) Update the switch and router route forwarding table(s) to effectively remove this endnode as a target for packets associated with this multicast group.

b) Remove the member from the group.

3.5.11.2.4 MULTICAST GROUP DELETE

When an (administrative) application deems there is no need for a multicast group or there are no other endnodes participating in a multicast group, the multicast group may be deleted. Upon receiving the delete request, the multicast management entity takes the following steps:

1) Unmap the multicast LID from the multicast group address.

2) Inform each router within this subnet that this subnet is no longer participating in the associated multicast group.

3.5.11.3 MULTICAST PRUNE

To improve fabric efficiency, the multicast group management entity should periodically verify that all endnodes and routers participating within a multicast group are still participating and if they are not, it should prune them from the multicast group by performing the multicast group leave algorithm. The verification period is outside the scope of IBA.

3.5.12 VERBS

IBA describes the service interface between a host channel adapter and the operating system by a set of semantics called *Verbs*. Verbs describe operations that take place between a host channel adapter and its operating system based on a particular queuing model for submitting work requests to the channel adapter and returning completion status.

The intent of Verbs is not to specify an API, but rather to describe the interface sufficiently permitting OS venders to define appropriate APIs that take advantage of the architecture.

Verbs describe the parameters necessary for configuring and managing the channel adapter, allocating (creating and destroying) queue pairs, configuring QP operation, posting work requests to the QP, getting completion status from the completion queue.

3.6 CHANNEL & MEMORY SEMANTICS

IBA communications provide the user with both channel semantics and memory semantics since both are useful for I/O and IPC. Channel semantics, sometimes called Send/Receive, refers to the communication style used in a classic I/O channel – one party pushes the data and the destination party determines the final destination of the data. The message transmitted on the wire only names the destination's QP, the message does not describe where in the destination consumer's memory space the message content will be written.

With memory semantics the initiating party directly reads or writes the virtual address space of a remote node. The remote party needs only communicate the location of the buffer; it is not involved with the actual transfer of the data.

A typical I/O transaction might use a combination of channel and memory semantics. For example, a host process might initiate an I/O operation by using channel semantics to send a disk write command to an I/O device. The I/O device examines the command and uses memory semantics to read the data buffer directly from the memory space of the processor

node. After the operation is completed, the I/O unit then uses channel semantics to push an I/O completion message back to the processor node.

3.6.1 COMMUNICATION INTERFACE

"*Channel adapter*" is the term that identifies the hardware that connects a node to the IBA fabric (and includes any supporting software). The channel adapter for a processor node is called a "*host channel adapter"* (HCA) and a channel adapter in an I/O node is a "*target channel adapter*" (TCA). A consumer communicates through one or more "queue pairs" (QP). An HCA typically supports hundreds or thousands of QPs while a TCA might support less than ten QPs.

It is the QP that is the communication interface. The user initiates work requests (WR) that causes work items, called WQEs, to be placed onto the queues and the channel adapter executes the work item.

Specifically, the operations supported for Send Queues are:

- **Send Buffer** -- a channel semantic operation to push a local buffer to a remote QP's receive buffer. The Send WR includes a gather list to combine data from several virtually contiguous local buffer segments into a single message that is pushed to a remote QP's Receive Buffer. The local buffer's virtual addresses must be in the address space of the consumer that created the local QP.
- **RDMA Read** -- a memory semantic operation to read a virtually contiguous buffer on a remote node. The RDMA Read operation reads a virtually contiguous buffer on a remote endnode and writes the data to a local memory buffer.

  Like the Send operation, the local buffer must be in the address space of the consumer that created the local QP.

  The remote buffer must be in the address space of the remote consumer owning the remote QP targeted by the RDMA Read.
- **RDMA Write** -- a memory semantic operation to write a virtually contiguous buffer on a remote node. The WR contains a gather list of local buffer segments and the virtual address of the remote buffer into which the data from the local buffer segments are written.

  Like the Send WR, the local buffer must be in the address space of the consumer that created the local QP.

  The remote buffer must be in the address space of the remote consumer owning the remote QP targeted by the RDMA Write.
- **Atomic** -- a memory semantic operation to do an atomic operation on a remote 64 bit word. The Atomic operation is a combined Read, Modify, and Write operation.

An example of an Atomic operation is the Compare and Swap if Equal operation. The WR specifies a remote memory location, a compare value, and a new value. The remote QP reads the specified memory location, compares that value to the compare value supplied in the message, and only if those values are equal, then the QP writes the new value to that same memory location. In either case the remote QP returns the value it read from the memory location to the requesting QP. The other atomic operation is the FetchAdd operation where the remote QP reads the specified memory location, returns that value to the requesting QP, adds to that value a value supplied in the message, and then writes the result to that same memory location.

- **Memory Bind** -- a memory management operation that changes the binding of a memory window. The Bind Memory Window operation associates a previously allocated Memory Window to a specified address range within an existing Memory Region, along with a specified set of remote access privileges.

For Receive Queues, there is only a single type of WR:

- **Post Receive Buffer** -- a channel semantic operation describing a local buffer into which incoming Send messages are written. The WR includes a scatter list describing several local buffer segments. The contents of an incoming Send message is written to these buffer segments in the order specified. The buffer's virtual addresses must be in the address space of the consumer that created the local QP. A WR without a scatter-gather list may be used to receive Immediate Data from a Write or a zero length Send operation.

Zero processor copy data transfer, with no kernel involvement, is key in providing high bandwidth, low latency communication. A consumer can transfer a data buffer via the QP directly from where the buffer resides in memory. Furthermore, the protection provided by R_Keys & L_Keys (memory registration) removes the need for the OS to validate data transfers. Thus the OS may allow posting the WQE from user-mode, bypassing the operating system, and thus consuming fewer instruction cycles.

IBA operations support the use of virtual addresses and existing virtual memory protection mechanisms to assure correct and proper access to all memory. Thus IBA applications are not required to use physical addressing for any operation.

A consumer can use either of two mechanisms to enable remote access to its memory (RDMA). First, when registering its memory (creating a Memory Region), the consumer can simply enable remote access for the entire Memory Region. If more control of remote access is desired, the consumer can allocate a Memory Window and bind it to a subset of an existing Memory Region. Either approach results in an R_Key. The con-

sumer then provides that R_Key and the virtual address of the data buffer to a remote node for use in subsequent RDMA operations. Only an incoming RDMA request with a correct R_Key can gain access to the specific area of memory. Furthermore, the QP and the Memory Region or Window must be in the same protection domain.

3.6.2 IBA TRANSPORT SERVICES

The IBA transport mechanisms provide multiple classes of communication services. When a QP is created, it is configured to provide one of these classes of transport services:

- **Reliable Connection** (acknowledged - connection oriented)
- **Reliable Datagram** (acknowledged - multiplexed)
- **Unreliable Connection** (unacknowledged - connection oriented)
- **Unreliable Datagram** (unacknowledged - connectionless)
- **Raw Datagram** (unacknowledged - connectionless)

The **Reliable Connection** service associates a local QP with one and only one remote QP. Thus a Send Buffer WQE placed on one QP causes data to be written into the Receive Buffer of the associated QP. RDMA operations operate on the address space of the associated QP.

A connected service requires each consumer to create a QP for each consumer with which it wishes to communicate. Thus if there are M consumers on each of N platforms that all wish to communicate via connected class of service, then each platform requires M^2 * N QPs. This assumes that each consumer on a particular platform communicates with consumers (including itself) on that same platform by taking advantage of the ability to connect a QP to a QP on the same node.

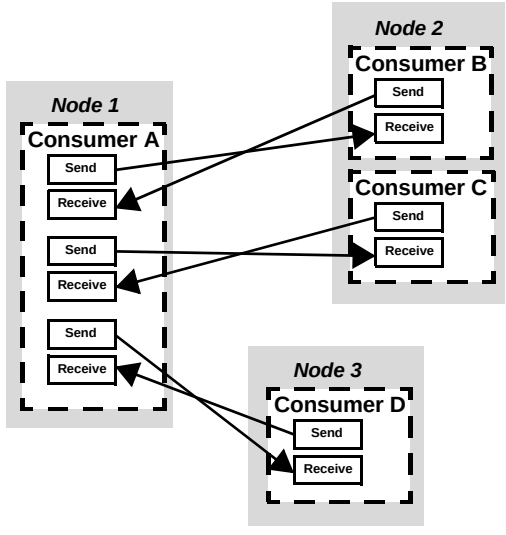


Figure 22 Connected Service

The Reliable Connection is reliable because the channel adapter can maintain sequence numbers and acknowledges all messages. A combination of hardware and channel adapter software retries any failed communications. The consumer of the QP sees reliable communications even in the presence of bit errors, receive buffer under runs, network congestion, and if alternate paths exist in the fabric, failures of fabric switches or links.

The acknowledgments ensure data is delivered reliably between the associated QPs and thus between each node's memory.

The acknowledgment is not a consumer level acknowledgment -- it doesn't validate that the receiving consumer has consumed the data. The acknowledgment only means the data has reached the destination.

The **Unreliable Connection** service also associates a local QP with one and only one remote QP. Thus a Send Buffer request placed on one QP causes data to be written into the Receive Buffer of the associated QP. RDMA Write operations operate on the address space of the associated QP.

Unlike reliable connection service, unreliable connection does not acknowledge and thus does not have the ability to resend lost or corrupted messages. Rather, lost or corrupted messages are simply dropped. Since there is no acknowledgment, RDMA Reads and Atomic operations are not supported. Because packets of an RDMA Write might be lost or corrupted, partial writing of a buffer might take place, but once a missing or corrupted packet is received, the write operation ceases until the start of a new message.

The **Unreliable Datagram** service is connectionless and unacknowledged. It allows the consumer of the QP to communicate with any unreliable datagram QP on any node. Receive operation allows incoming messages from any unreliable datagram QP on any node.

The Unreliable Datagram service greatly improves the scalability of IBA. Since the service is connectionless, an endnode with a fixed number of QPs can communicate with far more consumers and platforms compared to the number possible using the Reliable Connection and Unreliable Connection service.

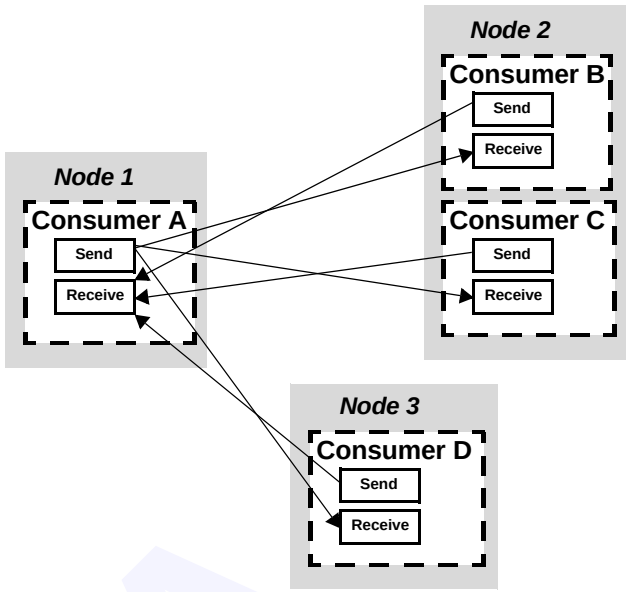


Figure 23 Datagram Service

This is the class of service used by management to discover and integrate new IBA switches and endnodes. It does not provide the reliability guarantees of the other service classes, but operates with less state maintained at each endnode. Unlike other services where messages are conveyed by multiple packets, the maximum message length is constrained in size to fit in a single packet.

The **Reliable Datagram** (RD) service is multiplexed over connections between nodes called End-to-end contexts (EEC) which allows each RD QP to communicate with any RD QP on any node with an established EEC. Multiple QPs can use the same EEC and a single QP can use multiple EECs (one EEC for each remote node per reliable datagram domain).

A reliable datagram domain (RDD) determine which sets of RD QPs can access which sets of EECs. Some possible reasons for multiple RDDs are traffic in different partitions, different QoS characteristics, security, and performance.

The EEC uses sequence numbers and acknowledgments associated with each message to ensure the same degree of reliability as with the Reliable Connection service. Each channel adapter maintains end-to-end specific state for each node to keep track of the sequence numbers, acknowledgments, and time-out values. Each EEC is shared by all Reliable Datagram QPs for that RDD.

For reliable datagram service on a per RDD basis, each consumer needs only to create a single QP and the node creates an EE context for each

platform with which it communicates. Thus if there are M consumers on each of N platforms that all wish to communicate via IBA reliable datagram service, then each platform requires M QPs and N end-to-end contexts.

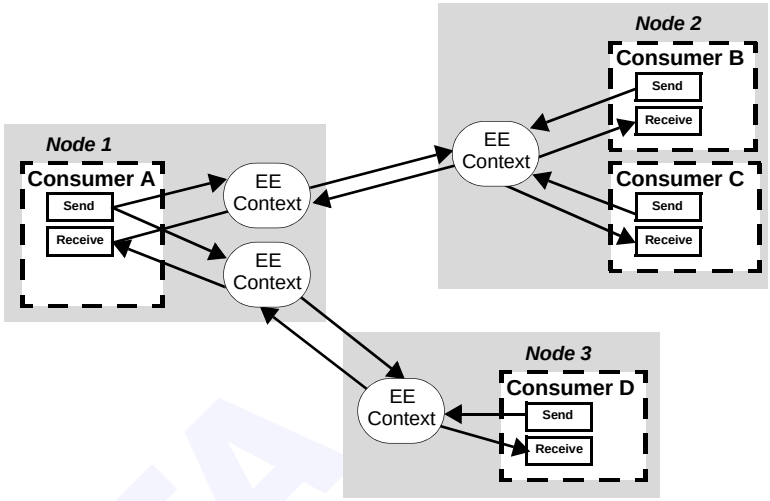


Figure 24 Reliable Datagram Service

The **Raw Datagram** service is not technically a transport but rather it is a data link service that allows a QP to send and receive raw datagram messages. There are two types of raw datagram service (EtherType and IPv6). The EtherType raw datagram packet contains a generic transport header that is not interpreted by the channel adapter, but it specifies the protocol type. The IPv6 raw datagram contains a global route header that identifies the protocol type.

Using IPv6 raw datagram service, the IBA channel adapter can support standard protocols layered atop IPv6, such as TCP and UDP. Thus native IPv6 packets can be bridged into the IBA SAN and delivered directly to a port and to its IPv6 raw datagram QP. This allows the raw datagram QP consumer to support multiple transport protocols.

Using EtherType raw datagram service, the IBA channel adapter can support standard protocols the same as Ethernet, including TCP and UDP as well as IPv4. Thus native ethernet packets can be bridged into the IBA subnet and delivered directly to a port and to its EtherType raw datagram QP.

When the QP is created, the consumer registers with the channel adapter in order to direct received datagrams to it (one QP for IPv6 and one for EtherType).

3.7 IBA LAYERED ARCHITECTURE

IBA operation can be described as a series of layers. The protocol of each layer is independent of the other layers. Each layer is dependent on the service of the layer below it and provides service to the layer above it.

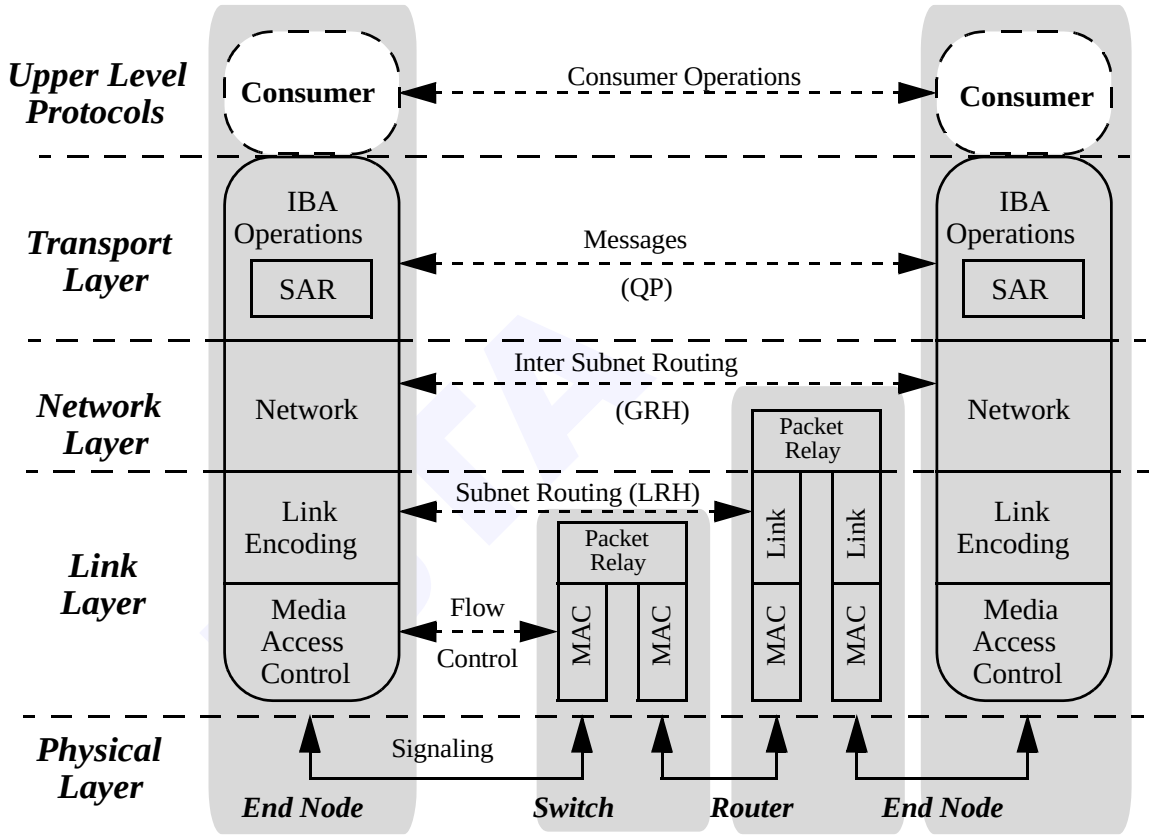


Figure 25 IBA Architecture Layers

3.7.1 PHYSICAL LAYER

The physical layer specifies how bits are placed on the wire to form symbols and defines the symbols used for framing (i.e., start of packet & end of packet), data symbols, and fill between packets (Idles). It specifies the signaling protocol as to what constitutes a validly formed packet (i.e., symbol encoding, proper alignment of framing symbols, no invalid or non-data symbols between start and end delimeters, no disparity errors, synchronization method, etc.)

.

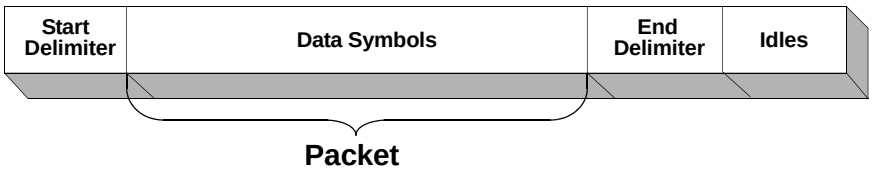


Figure 26 IBA Packet Framing

The physical layer specification is in Volume 2. It specifies the bit rates, media, connectors, signaling techniques, etc.

3.7.2 LINK LAYER

The link layer describes the packet format and protocols for packet operation, e.g. flow control and how packets are routed within a subnet between the source and destination. There are two types of packets.

- **Link Management Packet** - these are packets used to train and maintain link operation. These packets are created and consumed within the Link Layer and are not subject to flow control. Link management packets are used to negotiate operational parameters between the ports at each end of the link such as bit rate, link width, etc. They are also used to convey flow control credits and maintain link integrity. Link management packets are never forwarded to other links.
- **Data Packet** - these are the packets that convey IBA operations and they consist of a number of different headers, which might or might not be present.

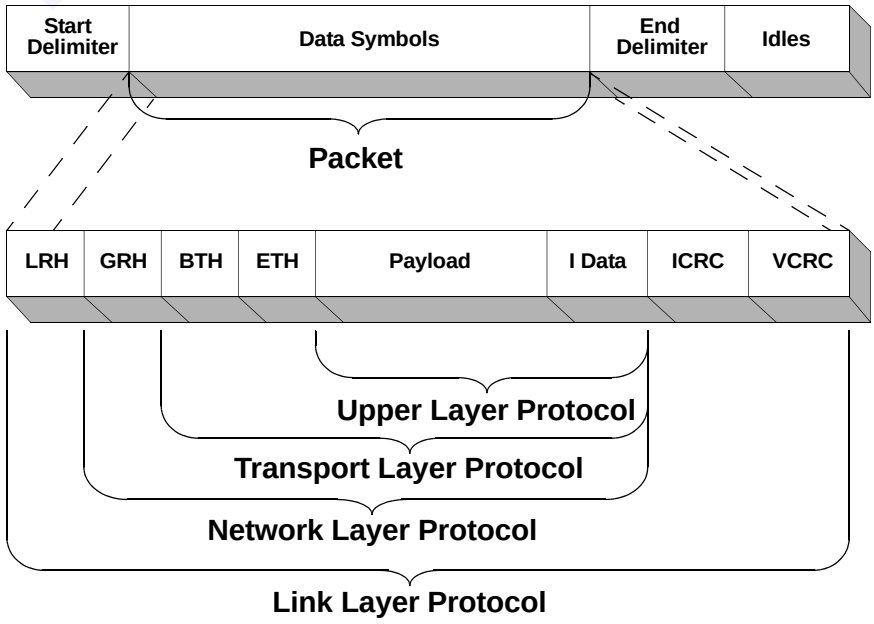


Figure 27 IBA Data Packet Format

The **Local Route Header** (LRH) is always present and it identifies the local source and local destination ports where switches will route the packet and also specifies the *Service Level* (SL) and VL on which the packet travels. The VL is changed as the packet traverses the subnet but the other fields remain unchanged.

The subnet manager assigns unique LIDs to each port of a channel adapter as well as the management entity of a switch. The source places the LID of the destination in the LRH and switches route the packet to that destination. If the packet is to be routed to another subnet, the packet's destination LID contains the LID of a router, otherwise the packet's destination LID specifies a LID assigned to a channel adapter (or switch, for certain of management packets).

There are two CRCs in each packet. The **Invariant CRC** (ICRC) covers all fields which should not change as the packet traverses the fabric. The **Variant CRC** (VCRC) covers all of the fields of the packet. The combination of the two CRCs allow switches and routers to modify appropriate fields and still maintain an end to end data integrity for the transport control and data portion of the packet. The coverage of the ICRC is different depending on whether the packet is routed to another subnet (i.e. contains a global route header).

Link level flow control is a credit based method where the receiver on each link sends credits to the transmitter on the other end of the link. Credits are per VL and indicate the number of data packets that the receiver can accept on that VL. The transmitter does not send data packets unless the receiver indicates it has room. VL15 (the management VL) is not subject to flow control.

3.7.3 NETWORK LAYER

The network layer describes the protocol for routing a packet between subnets.

The **Global Route Header** (GRH) is present in a packet that traverses multiple subnets. The GRH identifies the source and destination ports using GID in the format of an IPv6 address. Routers forward the packet based on the content of the GRH. As the packet traverses different subnets, the routers modify the content of the GRH and replace the LRH. But the source and destination GIDs do not change and are protected by the ICRC field. Routers recalculate the VCRC but not the ICRC. This preserves end to end transport integrity.

Each subnet has a unique subnet ID, the Subnet Prefix. When combined with a Port GUID, this combination becomes a port's Global ID (GID). A

node might have other locally administrated Global IDs. The source places the GID of the destination in the GRH and the LID of the router in the LRH. Each router forwards the packet through the next subnet to another router until the packet reaches the target subnet. The last router replaces the LRH using the LID of the destination.

3.7.4 TRANSPORT LAYER

The network and link protocols deliver a packet to the desired destination. The transport portion of the packet delivers the packet to the proper QP and instructs the QP how to process the packet's data.

The transport layer is responsible for segmenting an operation into multiple packets when the message's data payload is greater than the *maximum transfer unit* (MTU) of the path. The QP on the receiving end reassembles the data into the specified data buffer in its memory.

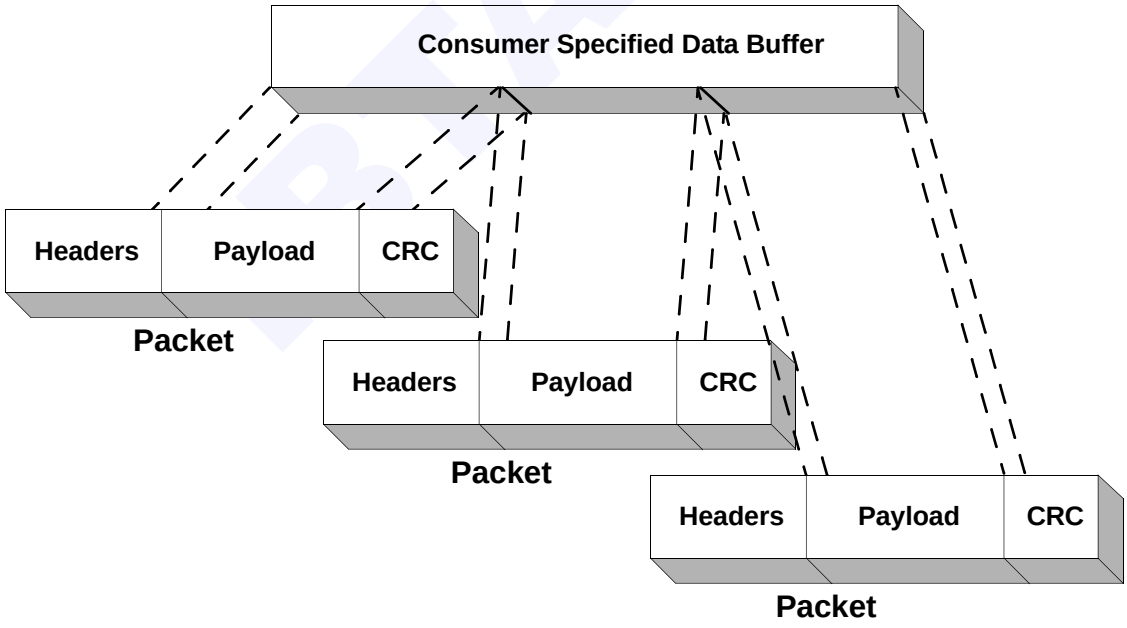


Figure 28 Segmentation of Data

The **Base Transport Header** (BTH) is present in all packets except for RAW datagrams. It specifies the destination QP and indicates the operation code, packet sequence number, and partition.

The operation code identifies if the packet is the first, last, intermediate, or only packet of a message and specifies the operation (Send, RDMA Write, Read, Atomic).

The packet sequence number (PSN) is initialized as part of the communications establishment process and increments each time the QP creates

a new packet. The receiving QP tracks the received PSN to determine if it lost a packet. For reliable service, the receiver sends an ACK or NAK packet back to notify the sender that packets were or were not received correctly. In this case the recipient discards subsequent packets until the sender resends the missing messages. For unacknowledged service, when the recipient detects a missing packet, it aborts the current operation and discards all subsequent packets until it receives one that specifies a first or only operation code. Then operation continues.

There are various **Extended Transport Headers** (ETH) conditionally present depending on the class of service and the operation code.

For reliable datagram service, the ETH identifies the **EE context** that the QP uses to detect missing packets.

The first message of an RDMA Read or Write operation contains an **RDMA ETH** that specifies the virtual address, R_Key, and total length of the data buffer to read or write. Subsequent RDMA write packets provide the remainder of the data. The QP validates that the memory is properly registered for access by that QP and that the total data written does not overrun the length specified. For an RDMA Read operation, the QP fetches the data, segments it into Read Response packets and sends them to the originator. When receiving a RDMA response, the QP writes the data into the buffer specified in the WQE of the RDMA Read Request.

An Atomic operation contains an **Atomic ETH** that specifies the virtual address and R_Key of the memory location that is the object of the operation as well as 2 operands. The QP validates that the memory is properly registered for access by that QP. The QP fetches the data, returns that value to the originator, performs the operation, and writes the result back to memory. For the Compare & Swap operation, the QP compares the content of the memory location with the first operand, and if they match, then it writes the second operand to that same location. Otherwise it does not modify it. For the Fetch & Add operation, the QP performs an unsigned add using the 64-bit Add Data field in the Atomic ETH, and writes the result back to the same memory location. In either case, operation is atomic such that another QP is not allowed to modify that memory location between the time of the read and the subsequent write.

The **Immediate Data** (IMMDT) field is optionally present in RDMA WRITE and SEND messages. It contains data that the consumer placed in the Send or RDMA Write request and the receiving QP will place that value in the current receive WQE. An RDMA Write with immediate data will consume a receive WQE even though the QP did not place any data into the receive buffer since the IMMDT is placed in a CQE that references the receive WQE and indicates that the WQE has completed.

For reliable connection service, IBA defines an end-to-end message level flow control. This allows the receiver to send credits to the transmitter as WQEs are posted to the receive queue. The QP tracks the number of WQEs posted and retired from the receive queue and keeps track of the number of messages received. It adds these numbers together to achieve a message limit value which it sends to the transmitter on the other end of the connection. The transmitter keeps track of the total number of messages that it creates and stops transmitting when it reaches the limit value established by the other end of the connection.

3.7.5 UPPER LAYER PROTOCOLS

As illustrated in Figure 29, IBA supports any number of upper layer protocols by various user consumers. IBA also defines messages and protocols for certain management functions. These management protocols are separated into Subnet Management and Subnet Services. Both of these have unique properties.

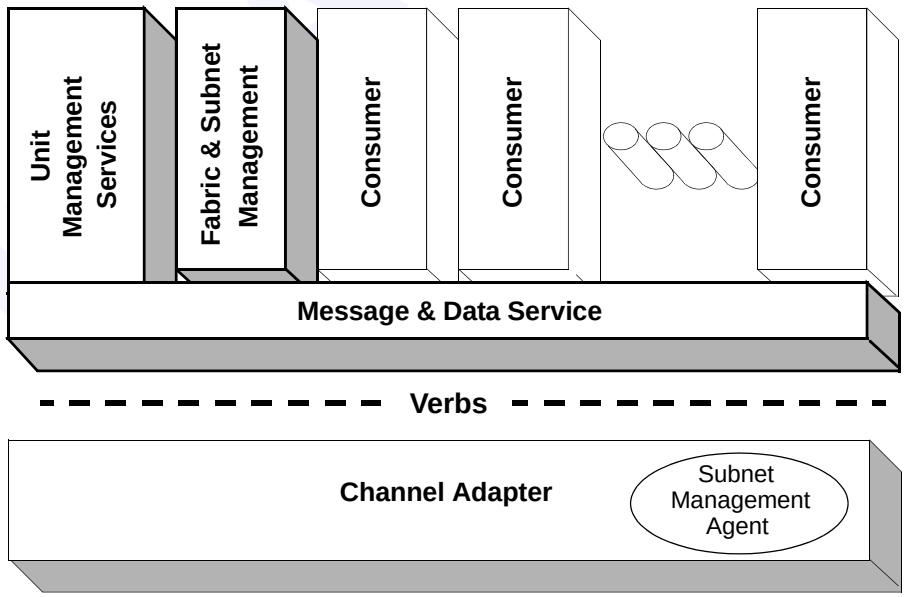


Figure 29 Upper Layers

3.7.5.1 SUBNET MANAGEMENT

Subnet Management is actually divided between the Subnet Manager (SM) application and the Subnet Management Agent (SMA). There only needs to be one subnet manager per subnet and it can reside in any node including switches and routers. Subnet management uses a special class of *Management Datagram* (MAD) called a *Subnet Management Packet* (SMP) which is directed to a special queue pair (QP0). As illustrated in Figure 30, each port has a QP0, and each node contains an SMA that:

- processes *Get()* and *Set()* SMPs received on QP0
- sends *GetResp()* SMPs out QP0
- sends *Trap()* SMPs out QP0.

A subnet manager:

- sends SMPs out QP0 to any port's QP0
- processes all SMPs received on QP0 except SMPs which are processed by that node's SMA.

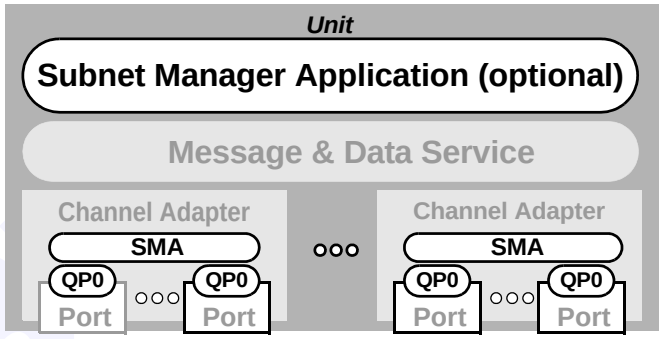


Figure 30 Subnet Management Elements

3.7.5.2 GENERAL SERVICES

General Service Agents (GSA*) actually consists of a number of management service agents as illustrated in Figure 31. Some of the services are optional. General services use a message format called a *General Management Packet* (GMP) which is a *Management Datagram* (MAD) and is normally directed to a special queue pair (QP1) called the General Service Interface (GSI). As illustrated in Figure 31, each port has a QP1, and all GMPs received on QP1 are processed by the one of the GSAs. The GSA is actually able to redirect GMPs for its particular class of service to another queue pair, allowing each GSA to maintain its own communication interface.

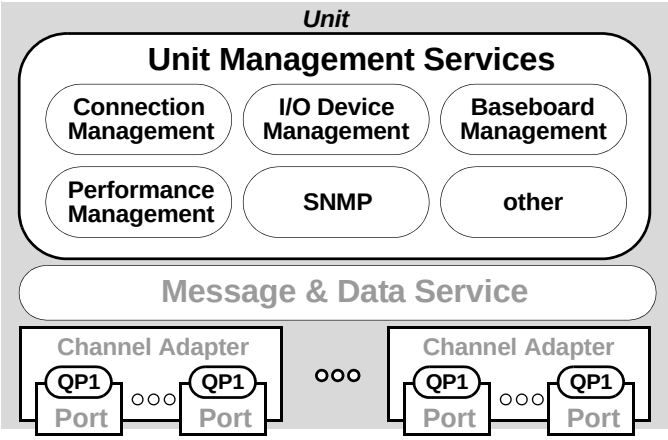


Figure 31 General Services

3.8 IBA TRANSACTION FLOW

A consumer interacts with an IBA channel adapter through a data structure called the Queue Pair, consisting of a Send Queue and a Receive Queue. A message is initiated by posting a work request which results in a WQE being placed on the Send Queue.

The channel adapter detects the WQE posting and accesses the WQE. The channel adapter interprets the command, validates the WQE's virtual addresses, translates it to physical addresses, and accesses the data. The outgoing message buffer is split into one or more packets. To each packet the channel adapter adds a transport header (sequence numbers, opcode, etc.). If the destination resides on a remote subnet the channel adapter adds a network header (source & destination GIDs). The channel adapter then adds the local route header and calculates both the variant and invariant checksums.

The flow of packets is subject to the link-level protocol over each link.

A packet is the unit of information that is routed through the IBA fabric. The packet is an endnode-to-endnode construct, in that it is created and consumed by endnodes. As the packet passes through switches, the switch may need to change the virtual lane and thus must replace the variant CRC with a new value but it does not touch the invariant CRC. If the packet passes through a router, the router changes the local route header and updates fields in the global route header, again updating the variant CRC but not changing the invariant CRC. Each switch and router moves the packet closer to its ultimate destination.

When a packet arrives at its final destination it goes through normal validity checks (e.g., framing violations, disparity, illegal characters, alignment, etc.) and both VCRC and ICRC are checked for integrity. The transport header identifies the target QP and the channel adapter uses context from that QP to validate that the packet came from the correct source, etc. and checks that the packet sequence number is valid (no missed packets). For a Send operation, the QP retrieves the address of the receive buffer from the next WQE on its receive queue, translates it to physical addresses, and accesses memory writing the data. If this is not the last packet of the message, the QP saves the current write location in its context and waits for the next packet at which time it continues writing the receive buffer until it receives a packet that indicates it is the last packet of the operation. It then updates the receive WQE, retires it, and sends an acknowledge message to the originator.

For reliable service types, if the QP detects one or more missing packets, it sends a NAK message to the originator indicating its next expected sequence number. The originator can then resend starting with the expected packet.

When the originator receives an acknowledgment, it creates a CQE on the CQ and retires the WQE from the send queue.

A QP can have multiple outstanding messages at any one time but the target always acknowledges in the order sent, thus WQEs are retired in the order that they are posted.

3.9 IBA MANAGEMENT INFRASTRUCTURE

IBA management defines a common management infrastructure for

- Subnet Management - provides methods for a subnet manager to discover and configure IBA devices and manage the fabric.
- General management services
  - Subnet administration - provides nodes with information gathered by the SM and provides a registrar for nodes to register general services they provide.
  - Communication establishment & connection management between endnodes
  - Mechanisms to discover and manage I/O devices “behind” channel adapters
  - Configuration management - an authority for assigning I/O resources to hosts
  - Performance management - monitors and reports well-defined performance counters

- Baseboard management - provides for power & chassis management using IB-ML as defined in Volume 2
- SNMP Tunneling (SNMP) - provides method for sending and receiving information between management agents and management applications. This includes Simple Network Management Protocol (SNMP), Desktop Management Interface (DMI), and Common Information Model (CIM), as well as other standard and proprietary interfaces.

The subnet management physical and logical models are illustrated in Figure 32. The general service models are illustrated in Figure 33 and Figure 34.

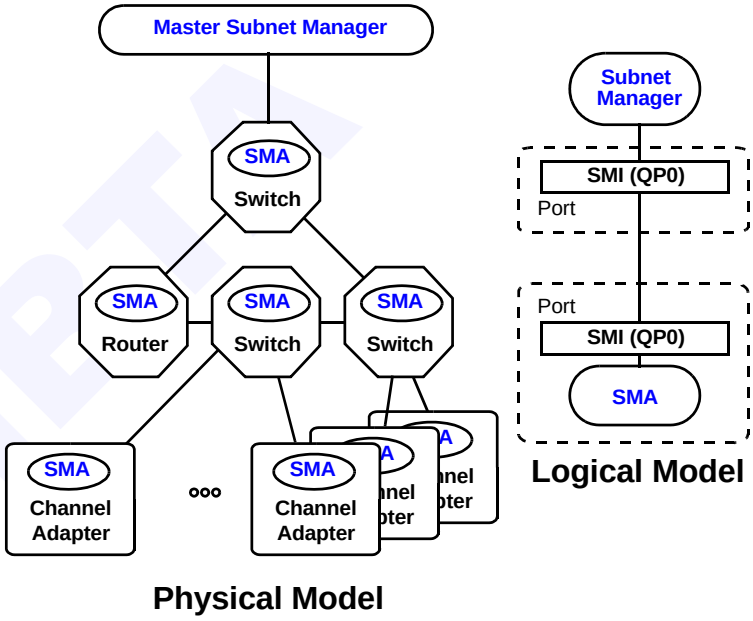


Figure 32 Subnet Management Models

Every channel adapter, switch, and router has a Subnet Management Agent (SMA) that responds to subnet management packets. Communication between the SM and SMAs use a well-known interface called the Subnet Management Interface (SMI) where each port has a QP with QP Number 0 (QP0) that is dedicated to sending and receiving SMPs.

Protection - The subnet manager can place a key (M_Key) in each node which can not be read by other nodes and prevents nodes without the M_Key from modifying a node's configuration. The SM only shares the M_Key with trusted peers as necessary. IBA also provides a lease expiration mechanism such that if the SM dies before it shares M_Key information with a successor, the lease expires, and the node returns to a state that allows the successor SM to establish a new M_Key.

IBA management defines the underlying interfaces and principles that allow IBA devices and the corresponding fabric to be discovered, initialized, and controlled. It defines a common management model and framework applicable to IBA-managed elements, identifies those elements, and defines their managed features. Management applications use this infrastructure to manage the IBA devices and communicate with other management applications.

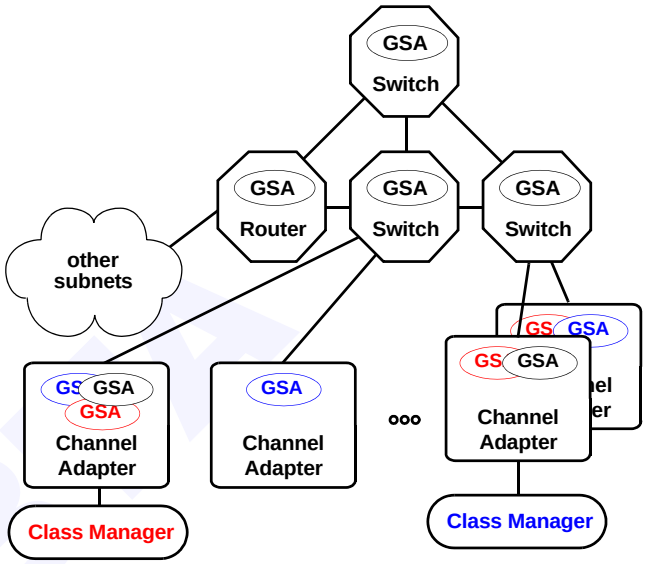


Figure 33 General Services Physical Model

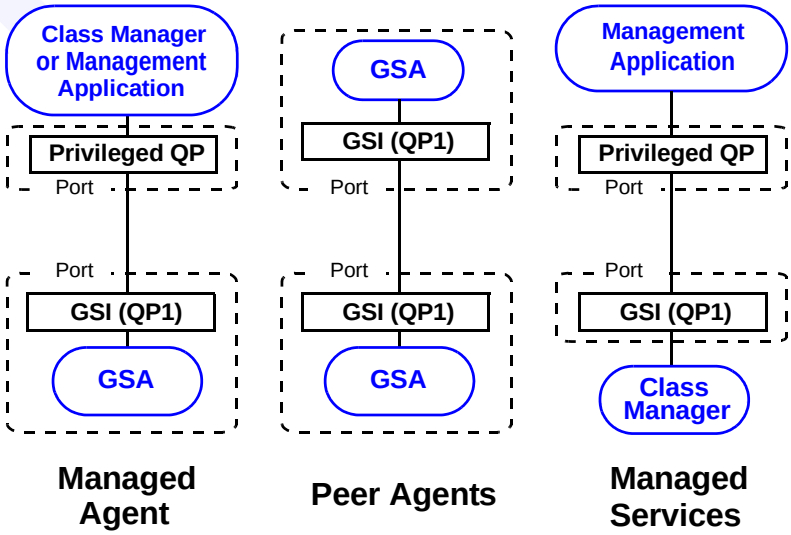


Figure 34 General Services Logical Models

IBA management infrastructure supports a number of different management service classes and logically provides for any node to host a class manager. Figure 34 illustrates different ways that management classes can use the management infrastructure.

- The Managed Agent model allows a class manager or management application to manage nodes through a General Service Agent (GSA) defined for that class present on each node to be managed. This is the same model used for subnet management and is the model used for I/O device management, baseboard management, SNMP, and performance management classes.
- The Peer Agents model allows managers resident on each node as a GSA to communicate with each other. This is the model used for communication management class.
- The Managed Service model allows management applications to access class managers. This is the model used for subnet administration and I/O resource management classes.

IBA management entails a variety of concepts, including:

- A means of configuring and gathering information from endnodes, switches and routers.
- A diagnostics framework as a common error handling mechanism.
- Installation and configuration services to allow for discovery and initialization of the fabric and endnodes.
- A standard management packet called a “Management Datagram” or “MAD”.
- Subnet Management Packets (SMP) as a subset of the MADs to allow set and get operations specifically between the Subnet Manager and IBA devices.
- General Management Packets (GMP) as the remaining subset of the MADs that allow management operations between the Subnet Manager and IBA devices and management operations between IBA devices themselves.
- Communication management services to allow setup and teardown of communications between channel adapters.
- Partitioning services to configure ports of an endnodes to be members of one or more possibly overlapping sets called partitions.

IBA provides the means for the operating system to restrict access to the management infrastructure. For the SMI, subnet management packets must be sourced from QP0. The GSI uses a privileged Q_Key (i.e., a Q_Key with the most significant bit set). Host channel adapters do not permit a privileged Q_Key to be specified in a work request, rather the QP must be configured for privileged operation by configuring the QP context with the privileged Q_Key. This permits management applications and class mangers to maintain their own QPs. The GSI uses QP1 for initial communication but allows traffic for a particular class to be redirected to a privileged QP.

3.9.1 MANAGEMENT DATAGRAMS

IBA defines a standard format for management messages which supports common processing. Each MAD contains the same header format that identifies the class of management message and the method. SMPs are one class of management message, another is directed route SMPs. MADs for other classes are called General Management Packets (GMPs) and include subnet administration, communication management, performance management, SNMP, device management, and baseboard management.

MADs begin with a standard header that carries information common to all classes. For classes that use the reliable multi-packet protocol (RMPP), this is immediately followed by an additional header for RMPP information. RMPP is a protocol that permits management entities to exchange more data than will fit in a single MAD. RMPP may be used by any management class. For example, it is used by the Subnet Administration class to give a client information about collections of paths between nodes.

3.9.2 MANAGEMENT METHODS

IBA defines common methods that may be adopted by any class. These include Get, Set, GetResp, Send, Trap, TrapRepress, Report, and ReportResp. Of course each management class defines their own set of attributes. These methods are sufficient for many classes but IBA also provides for class specific methods.

3.9.2.1 GETS & SETS

Gets and Sets are the most common use of MADs, especially the SM. The SM polls the fabric and learns its topology by sending SMP Get request messages. Each destination responds by sending a GetResp response message that includes the requested data. The SM configures IBA devices by sending Set request messages. This is effectively a Set & Get request. Each destination responds by sending a GetResp response message that includes the data values after the set action. Since not all parameters are settable or they might have limits, the originator inspects the response message to determine the true effect of the set request message.

3.9.2.2 TRAPS AND NOTICES

A trap is a message sent by a management agent to its class manger when certain asynchronous management events occur (such as protocol violations). Notices are attributes that can be queued in a Notice Queue at the managed node and may be retrieved and cleared by the class manager. The trap message has its data in the form of a notice attribute. The class manager programs the node with information about where to send

traps and the node stops sending the trap when it receives a TrapRepress from the class manager.

IBA devices use SMPs to send traps to the subnet manager when certain events occur. One such use is for a switch to send a trap to the subnet manager when it detects a state change on one of its ports (i.e., a topology change and/or device joining or leaving). Of course since SMPs are unreliable, the SM can not solely depend on this type of notification, but successful traps will decrease the latency in managing the topology change.

3.9.2.3 SENDS

Send() is a management method where one entity sends data to another on a class specific basis. Unlike Gets and Sets where the response contains the same attribute and transaction ID as the request, there is no explicit response to a Send. A management protocol based on Sends might respond with another Send, which may contain an entirely different attribute. In this case, information needs to be included in the attribute to correlate one Send to another.

3.9.2.4 REPORTS

The management infrastructure provides a means for management entities to subscribe to the class manager in order to have the class manager forward traps it receives from management agents it controls. A management entity subscribes by sending a Set to the class manager identifying the port and QP where the class manger is to forward the traps. When the class manager receives a trap, it forwards the notice attribute to the subscribed entity in the form of a Report MAD and the entity responds with a ReportResp MAD to let the class manager know that it received the Report. The class manager continues sending Reports until the subscribed entity responds with a ReportResp.

3.9.3 MANAGEMENT INTERFACES

IBA defines two well known QPs for management interfaces. **QP0** is reserved for subnet management and **QP1** is designated for general management services.

3.9.4 SUBNET MANAGEMENT INTERFACE

Every IBA port has a QP dedicated to subnet management. This is QP0. QP0 has special features that make it unique compared to other QPs.

- QP0 is permanently configured for Unreliable Datagram class of service.
- Each port of an IBA device has a QP0 that sends and receives packets.
- QP0 is a member of all partitions (i.e., can accept any packet specifying any partition).

- Only subnet management packets (SMPs) are valid
- Traffic for QP0 (i.e., SMPs) exclusively uses VL15, which is not subject to link-level flow control.

3.9.4.1 FABRIC INITIALIZATION

The subnet manager uses this service interface to poll and configure the fabric. Switches support a special routing mode known as *directed routing* that allows SMPs to be routed through switches prior to switches being configured with their forwarding database and prior to nodes being assigned local IDs. The subnet manager walks its way through the fabric sending SMPs to a device and discovering if it is a switch. Using directed routing, it can then send SMPs out each of the switch's ports to discover the devices connected to the switch. This process continues until the subnet manger discovers all of the devices and how they are interconnected.

Once the SM learns the subnet's topology, it configures each node with local IDs and configures the routing tables of switches. Once the fabric has been configured, SMPs can be sent using destination routing.

IBA allows multiple subnet mangers per subnet but only one can be the master manager. Thus IBA defines how a subnet manager detects the presence of another subnet manager and the arbitration mechanism for selecting which will be the master subnet manager.

3.9.4.2 DIRECTED ROUTES

A SMP can specify the route it takes through the fabric. This is done by including in the SMP a list of port numbers that define a path through the subnet (i.e., the path vector). The path vector specifies the output port for each switch along the path. The packet contains two path vectors (one for the forward route and one for the reverse route), a direction bit that indicates which path vector to traverse, and a hop pointer that indicates the current position in the path vector. The reverse path vector is built by switches as they process the forward path vector.

When a switch receives a directed routed SMP, it uses the current hop pointer to identify where in the path vector it is. If the direction is "forward" it determines the output port from the forward path vector, updates the reverse path vector by adding the port number on which it received the SMP, increments the current hop pointer, and then forwards the packet out the specified output port. When the packet reaches the destination, the target device uses the reverse route field for the reply by simply changing the sense of the direction bit and sending the reply SMP out the port on which it was received. Because the direction is "reverse" each switch now decrements the current hop pointer, uses it to determine the original input port, and then sends the packet out that port.

3.9.5 GENERAL SERVICE INTERFACE

Every IBA channel adapter has a QP dedicated to general fabric services. This is QP1. QP1 has special features that make it unique compared to other QPs.

- QP1 is permanently configured for Unreliable Datagram class of service.
- Each port of an IBA device has a QP1 that sends and receives packets.
- QP1 is a member of all of the port's partitions (i.e., can accept any packet specifying a P_Key contained in the port's P_Key table).
- Only management datagrams (MADs) are valid
- Traffic for QP1 does not use VL15

3.9.5.1 REDIRECTION

QP1 being a well known interface has its advantages and disadvantages. One disadvantage is that all management classes go into the same queue which tends to bottleneck and promote head of line blocking. Thus IBA defines a mechanism that allows the channel adapter to redirect general service requests to other QPs.

When a channel adapter receives a GMP on QP1, it may respond with a redirect response indicating a new port and QP. The originator then resends the request to the new address and also uses that address for all subsequent requests for that same management class.

3.10 I/O OPERATION

IBA I/O architecture supports a range of I/O implementation from simple native devices to massive I/O subsystems. The model for an I/O unit is shown in Figure 35. An I/O unit is composed of a channel adapter and a number of I/O controllers. The channel adapter of an I/O unit is referred to as a Target Channel Adapter (TCA). A TCA has the same functionality as the HCA, but unlike the HCA, it is not necessarily designed for generic use, which means that it only needs to support the capabilities required by its controllers.

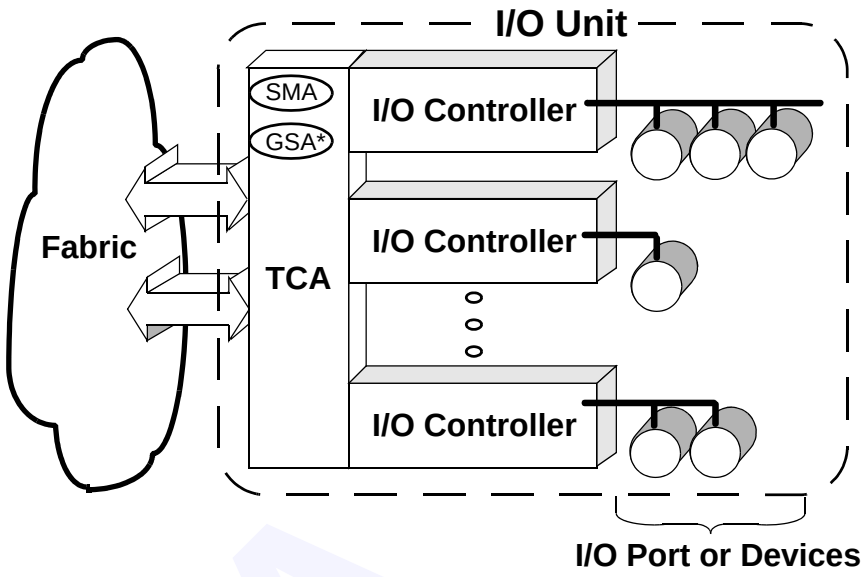


Figure 35 I/O Unit

I/O controllers represent the hardware and software that processes I/O transaction requests. Examples of I/O controllers are a SCSI interface controller, a RAID processor, a storage array processor, a LAN port controller, a disk drive controller, a console service.

The I/O unit contains a Subnet Management Agent (SMA) that responds to SMPs received on QP0. The I/O unit also contains general service agents (GSA*) that responds to GMPs received on the GSI (QP1). The GSA* contains at least Communication Management and I/O Device Management (DevMgt). Each I/O controller is registered with the DevMgt GSA such that it can respond to DevMgt GMPs with specific information about the controller.

Typically an I/O resource manager in the processor node sends DevMgt GMPs to an I/O unit to discover the attributes of the controllers. The attributes contain sufficient information for the I/O resource manager to identify the appropriate I/O driver. The I/O resource manager loads the driver, if necessary, and configures the I/O driver with the identity of the controller (IO Unit and Controller ID). The I/O driver then creates the appropriate communication ports (i.e., QPs) on the processor node and calls the processor node's communication manager to create the appropriate connections with the I/O controller. Once the connections are established, the I/O driver exchanges control messages and data over the connections.

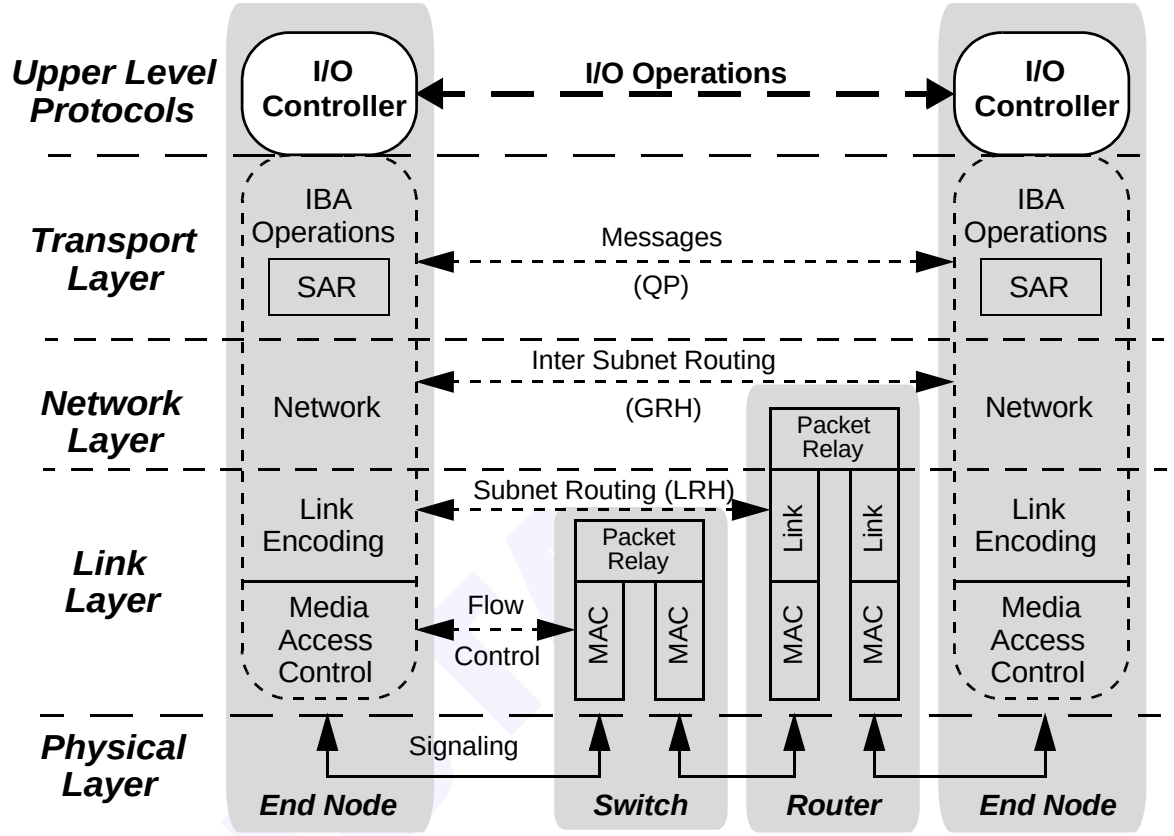


Figure 36 IO Operation

The number of communication ports used by the driver is an implementation variable. An I/O driver may use any available class of service (reliable connection, unreliable connection, reliable datagram, or unreliable datagram) and might use various classes of service for different communication ports.

CHAPTER 4: ADDRESSING

This chapter defines IBA addressing terminology and concepts. To facilitate understanding, refer to the following figures.

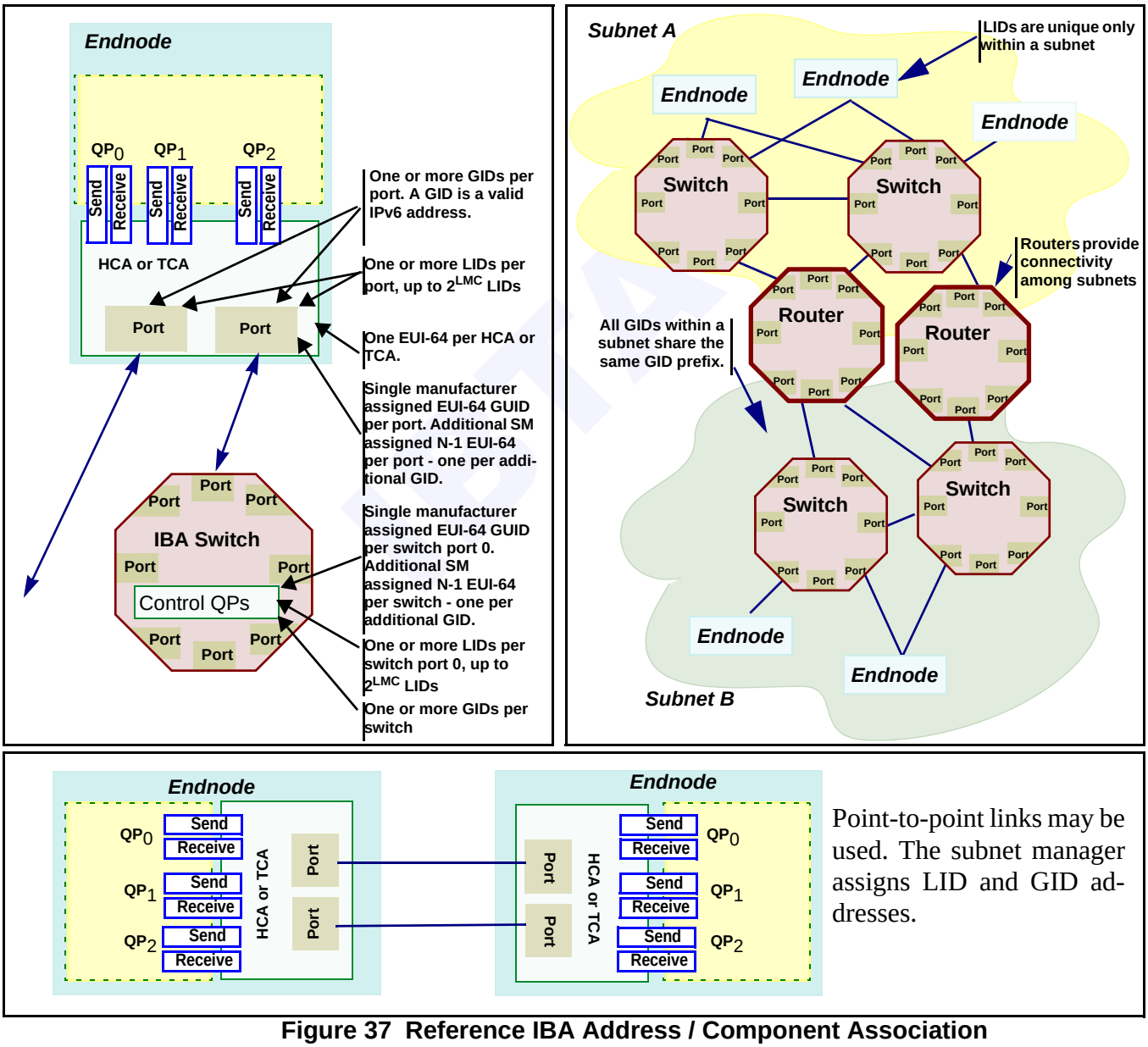


Figure 37 Reference IBA Address / Component Association

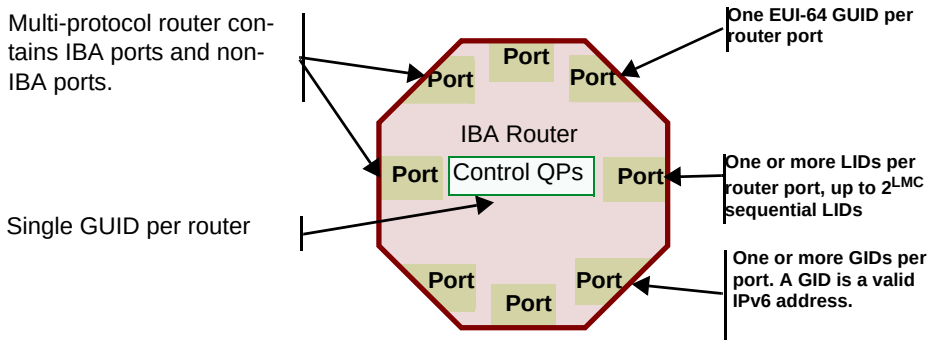


Figure 38 Reference IBA Router Address Association

4.1 TERMINOLOGY AND CONCEPTS

Endport: An endport is a CA port, a router port, or a switch management port.

Unicast Identifier: An identifier for a single endport. A packet sent to an unicast identifier is delivered to the endport identified by that identifier. IBA defines two unicast identifiers - a global identifier (GID) - may be unique across subnets - and a local identifier (LID) - unique only within a subnet).

Multicast Identifier: An identifier for a set of endports. A packet sent to a multicast identifier is delivered to all endports identified by that identifier. IBA defines two multicast identifiers - a global identifier (GID) used by applications to address a multicast group and route packets between subnets and a local identifier (LID) used to switch packets within a subnet.

EUI-64: IEEE defined 64-bit identifier assigned to a device. The EUI-64 is a 64-bit identifier created by concatenating a 24-bit company_id value and a 40-bit extension identifier. The company_id is assigned by the IEEE Registration Authority; the extension identifier is assigned by the organization with the assigned company_id.

- The manufacturer assigns an EUI-64 with global scope set. A SM may assign additional EUI-64 with local scope indicated.
- For additional details, see: “Guidelines For 64-bit Global Identifier (EUI-64) Registration Authority”at www.standards.ieee.org/regauth/oui/tutorials/EUI64.html

GUID (Global Unique Identifier): A globally unique EUI-64 compliant identifier.

C4-1: Each HCA, TCA, switch, and router shall be assigned an EUI-64 GUID by the manufacturer.

C4-2: Each endport shall be assigned an EUI-64 GUID by the manufacturer.

Subnet Prefix: A 0 to 64-bit - as a function of scope - identifier used to uniquely identify a set of endports which are managed by a common subnet manager.

GID Prefix: A 64-bit identifier (upper 64-bits of a GID) created by concatenating address scope bits, potentially a small number of "filler" bits, and potentially a subnet prefix - filler and subnet prefix presence is a function of the address scope.

GID (Global Identifier): A 128-bit unicast or multicast identifier used to identify an endport or a multicast group. A GID is a valid 128-bit IPv6 address (per RFC 2373) with additional properties / restrictions defined within IBA to facilitate efficient discovery, communication, and routing. Note: These rules apply only to IBA operation and do not apply to raw IPv6 operation unless specifically called out.

C4-3: GIDs shall comply with the rules defined within 4.1.1 GID Usage and Properties on page 143:

4.1.1 GID USAGE AND PROPERTIES

1) Each endport shall be assigned at least one unicast GID. The first unicast GID assigned shall be created using the manufacturer assigned EUI-64 identifier. This GID is referred to as **GID index 0** and is formed by techniques 3(a) and 3(b) described below.
2) The default GID prefix shall be (0xFE80::0). A packet using the default GID prefix and either a manufacturer assigned or SM assigned EUI-64 must always be accepted by an endnode. A packet containing a GRH with a destination GID with this prefix must never be forwarded by a router, i.e. it is restricted to the local subnet.
3) A unicast GID shall be created using one or more of the following mechanisms:
    a) Concatenation of the default GID prefix with the manufacturer assigned EUI-64 identifier associated with an endport. This GID is referred to as the default GID.
    b) Concatenation of a subnet manager assigned 64-bit GID prefix and the manufacturer assigned EUI-64 identifier associated with an endport.
    c) Assignment of a GID by the subnet manager. The subnet manager creates a GID by concatenating the GID prefix (default or assigned) with a set of locally assigned EUI-64 values (at GID index 1 or above).

Each endport must be assigned at least one unicast GID using (a). Additional GIDs may be assigned using (b) and/or (c). Note: A subnet shall only have one assigned GID prefix (non default) at any given time.

4) Any QP in a CA, switch or router shall be addressable using the default GID prefix in addition to the assigned GID for that QP. This allows a subnet to transition from a default GID prefix state to a managed state without interrupting existing communication sessions.

5) The maximum number (N) of unicast GIDs supported per endport is implementation specific. The subnet manager may assign N-1 additional unicast GIDs. Each of these N-1 GIDs is created by concatenating one subnet manager assigned EUI-64 identifier (the local bit set) with the GID prefix.

6) The unicast GID address 0:0:0:0:0:0:0:0 is reserved - referred to as the Reserved GID. It shall never be assigned to any endport. It shall not be used as a destination address or in a global routing header (GRH).

7) The unicast GID address 0:0:0:0:0:0:0:1 is referred to as the loopback GID and is only used by raw IPv6 services - it is not used by IBA transport services. It shall never be assigned to an endport or be present in any IBA packets.

8) The unicast GID subnet prefix shall be limited to the upper 64-bits of the GID address space. The number of subnet prefix bits may further be limited by filler and scope bits - see below.

9) The lower 64-bits of the unicast GID cannot be further partitioned into subnets.

10) The lower 64-bits of a unicast GID shall be subnet unique. If the universal / local bit is set to universal, then the assignment must be globally unique.

11) The GRH (Global Route Header) shall contain valid source and destination GIDs. For raw IPv6 packets, an IPv6 routing header shall contain source and destination addresses in compliance with RFC 2373.

12) Unicast GID scoping shall be:

    a) Link-local - A unicast GID used within a local subnet using the default GID prefix. Routers must not forward any packets with either link-local source or destination GIDs outside the local subnet. A link-local GID has the following format:

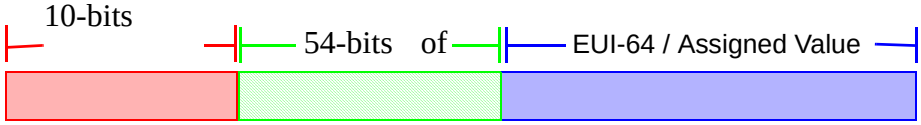


Figure 39 Link-Local Unicast GID Format

b) Site-local - A unicast GID used within a collection of subnets which is unique within that collection (e.g. a data center or campus) but is not necessarily globally unique. Routers must not forward any packets with either a site-local Source GID (SGID) or a site-local Destination GID (DGID) outside of the site.

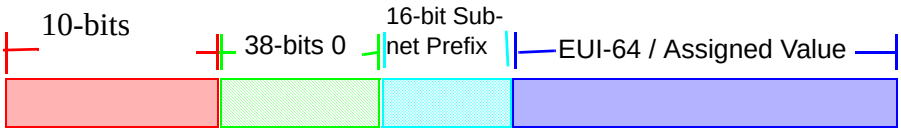


Figure 40 Site-Local Unicast GID Format

c) Global - A unicast GID with a global prefix, i.e. a router may use this GID to route packets throughout an enterprise or internet. The global GID format is:

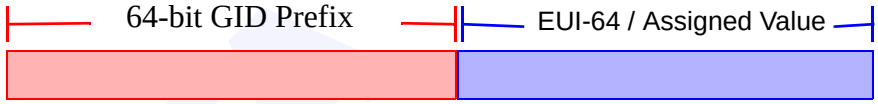


Figure 41 Unicast Global GID Format

13) A multicast group is uniquely identified by a multicast GID (MGID). Further, in addition to having the same MGID, all members of the multicast group must share the same P_Key and Q_Key.

14) The multicast GID format is:

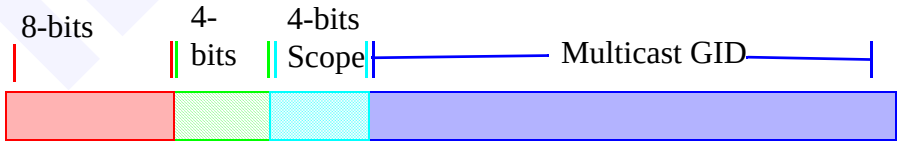


Figure 42 Multicast GID Format

a) 8-bits of 11111111 at the start of the GID identifies this as being a multicast GID.

b) Flags is a set of four 1-bit flags: 000T with three flags reserved and defined as zero ('0'). The T flag is defined as follows:

vi) T = 0 indicates this is a permanently assigned (i.e. well-known) multicast GID. See RFC 2373 and RFC 2375 as reference for these permanently assigned GIDs.

vii) T = 1 indicates this is a non-permanently assigned (i.e. transient) multicast GID.

c) Scope is a 4-bit multicast scope value used to limit the scope of the multicast group. The following table defines scope value and interpretation.

Table 3 Multicast Address Scope

Scope Value	Address Scope
0	Reserved
1	Unassigned
2	Link-local
3	Unassigned
4	Unassigned
5	Site-local
6	Unassigned
7	Unassigned
8	Organization-local
9	Unassigned
0xA	Unassigned
0xB	Unassigned
0xC	Unassigned
0xD	Unassigned
0xE	Global
0xF	Reserved

15) An endport may join zero, one or more multicast groups, i.e. an endport may be assigned zero, one or more multicast GIDs.

16) Multicast GIDs shall not appear as the source GID (SGID) in the GRH.

17) Multicast GID FF02:0:0:0:0:0:0:1 is the link-local multicast GID - a router should not route packets with this destination GID outside the local subnet. This GID is used as the destination address within the global router header (GRH) for communicating to a set of QPs participating within the all channel adapters multicast group - includes all channel adapters and enhanced switch port 0 endnodes that wish to participate in this group. ALL CHANNEL ADAPTERS MULTICAST GROUP is used to implement a broadcast service to all channel adapters which are capable of participating in multicast operations (must share the same MGID, P_Key, and Q_Key).

18) IPv6 defines a set of reserved multicast addresses in RFC 2375 and RFC 2373. IBA, unless explicitly stated otherwise, shall not use these addresses for IBA multicast operations and defines them as reserved for raw IPv6 usage.

4.1.2 CHANNEL ADAPTER, SWITCH, AND ROUTER ADDRESSING RULES

C4-4: Channel Adapters, Switches, and Routers shall comply with the addressing rules defined within 4.1.2 Channel Adapter, Switch, and Router Addressing Rules on page 147.

Addressing rules are:

1) A port shall attach to one link.
2) An endport shall support a range of LIDs as defined by a Base LID and an LMC. The LIDs shall be sequentially ordered starting with a base LID plus ($2^{LMC} - 1$) LIDs. The SM may program the LMC on an endport to any value between 0 and 7, to allow use of multiple LIDs (1-128) in addressing the endport.
   a) Base switch port 0 shall be assigned a single unicast LID, i.e. LMC = 0.
3) A unicast LID shall map to only one endport.
4) A multicast LID shall map to one or more endports - an endport may be the target of zero, one, or more multicast flows.
5) Unicast GIDs shall be assigned on a per endport basis.
6) A multiport CA (and by definition, a router) may be attached to one or more subnets - an endport shall only be attached to one subnet at a time.

4.1.3 LOCAL IDENTIFIERS

C4-5: Local Identifiers (LIDs) shall comply with the rules defined within 4.1.3 Local Identifiers on page 147.

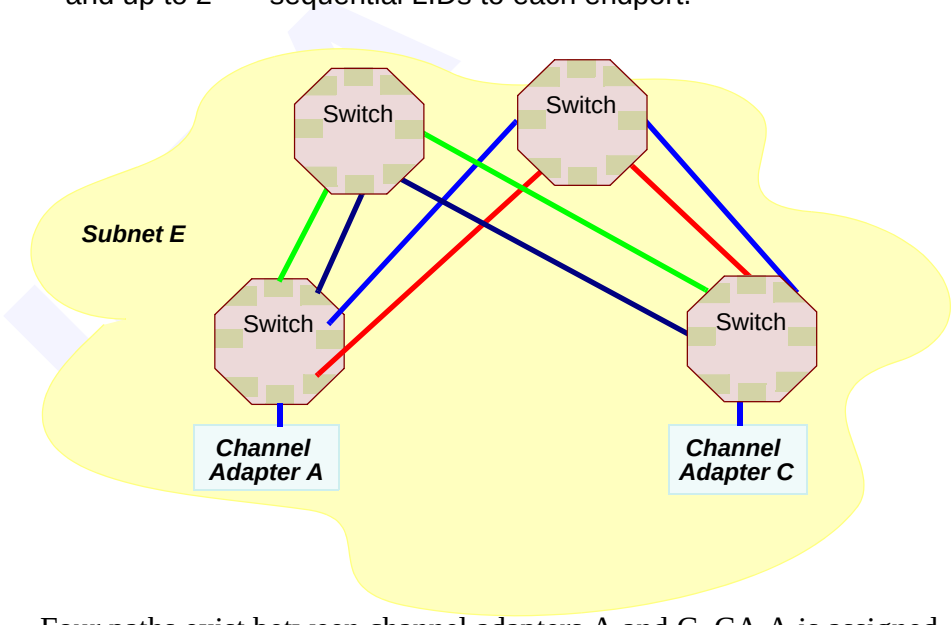
Local identifier (LID): A 16-bit identifier with the following properties:

1) A LID is assigned by the Subnet Manager (SM) and is subnet unique, i.e. it cannot be used to route between subnets.
2) The LID address space is divided into reserved, unicast and multicast address ranges.
3) LIDs are contained within the LRH (Local Route Header).
4) A source LID (SLID) shall refer to the endport that first injected the packet into the subnet.
5) A SLID shall only be associated with a unicast address.

6) A unicast destination LID (DLID) shall refer to the destination endport. A multicast DLID refers to the set of destination endports within the subnet participating in a given multicast group.

7) If the destination endport is not on the same subnet, the DLID shall refer to the router port responsible for forwarding the packet to the next hop to the destination endport.

8) From any point within a subnet, a given endport may receive packets through multiple physical paths within the subnet. Each physical path may be identified by one or more destination LIDs. To facilitate multipath operation while minimizing channel adapter complexity, each endport shall be assigned a base LID and a LID Mask Control (LMC) value by the subnet manager. The LMC is a 3-bit field which represents 2^{LMC} paths (maximum of 128 paths). During discovery, the subnet manager may determine the number of paths to a given endport and will partition the 16-bit LID space to assign a base LID and up to 2^{LMC} sequential LIDs to each endport.



Four paths exist between channel adapters A and C. CA A is assigned a Base LID 4, LMC = 2. This translates to CA A being assigned LIDs: {4, 5, 6, 7}. CA C is assigned Base LID 8, LMC = 2. This translates into CA C being assigned LIDs: {8, 9, 10, 11}.

Figure 43 Multipath Identification

Note: The base LID must have LMC least significant bits set to 0. For example, if the LMC = 0, the base LID may be any unicast LID. If the LMC = 7, the base LID set the 7 least significant bits to zero.

9) The LID space is defined as follows:

- LID 0x0000 is reserved.

- LID 0xFFFF is defined as a permissive DLID. The permissive DLID indicates that the packet is destined for QP0 on the endport which received it. LMC is not defined for this address.
- The unicast LID range is a flat identifier space defined as 0x0001 to 0xBFFF.
- The multicast LID range is a flat identifier space defined as 0xC000 to 0xFFFE.
    - The DLID for any packet which contains a multicast GID shall be within the above specified multicast LID range.

10) A multicast LID may be overloaded by multiple multicast GIDs, i.e. there may be a many-to-one MGID to MLID mapping within a given subnet. When a multicast LID is overloaded, the multicast groups sharing the same MLID must have the same P_Key. This simplification is required to allow switches and routers that implement optional P_Key enforcement for multicast operations.

CHAPTER 5: DATA PACKET FORMAT

This chapter introduces the fields in the data packet. A brief description of each field is given including a definition, field size, and abbreviation. This chapter does not specify the details of each field, but only the general usage and layout of the fields.

In addition to data packets, IBA defines link packets which are used for link-level flow control. The format of these link packets is described in 7.9.4 Flow Control Packet on page 210.

In this specification, the term packet refers to data packets only (i.e. packet and data packet are synonymous). Where reference to link packets is intended, the full term *link packet* will be used.

5.1 PACKET TYPES

Packets are the unit of transfer in IBA. As described in 3.3 Communications Stack on page 94 messages are segmented into packets by the CAs for transmission across the IB fabric.

Packets have the following attributes:

- Indivisible unit of data transfer and routing
- Unit of acknowledgement
- Unit of segmentation and re-assembly for messages
- Unit of link-level flow control

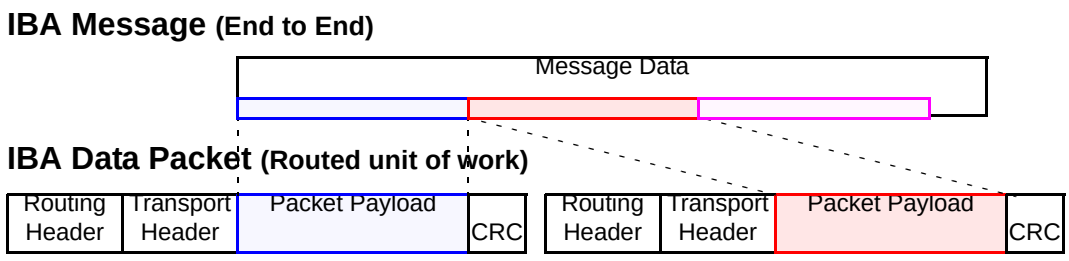


Figure 44 IBA Messages and Packets

There are two general classes of transports used in Packets:

- **IBA Packets** have IBA defined transport headers, are routed on IBA fabrics, and use native IBA transport facilities.

- **Raw Packets** may be routed on IBA fabrics but do not contain IBA transport headers. From the IB point of view, these packets contain only IBA routing headers, payload and CRC. IBA does not define the processing of these packets above the link and network layers. The intent is that these packets can be used to support non-IBA transports over an IB fabric.

5.2 DATA PACKET FORMAT

The overall data packet structure is shown in Figure 45 on page 152. There are two routing headers that precede a transport header(s) and payload:

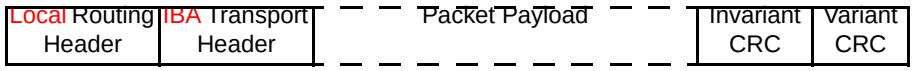
- The local route header is required on all packets
- The global route header is required on all packets that are to be routed to a different subnet, and on all multicast packets regardless of destination.
- A global route header may be placed on any packet except subnet management packets.

C5-1: Packets generated by an InfiniBand device shall conform to the packet structure defined in Figure 45 and to the packet header location and size requirements as defined in figure 46

Each IBA packet ends with an invariant CRC followed by a variant CRC.

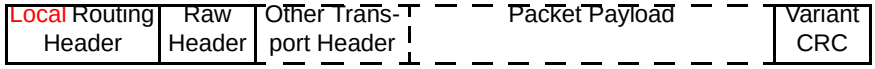
Each raw packet ends with a variant CRC.

Local (within a subnet) Packets



Global (routing between subnets) Packets

Raw Packet with Raw Header



Raw Packet with IPv6 Header

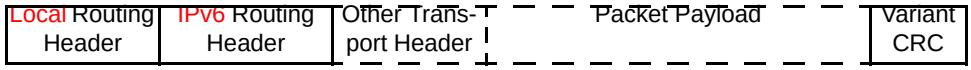
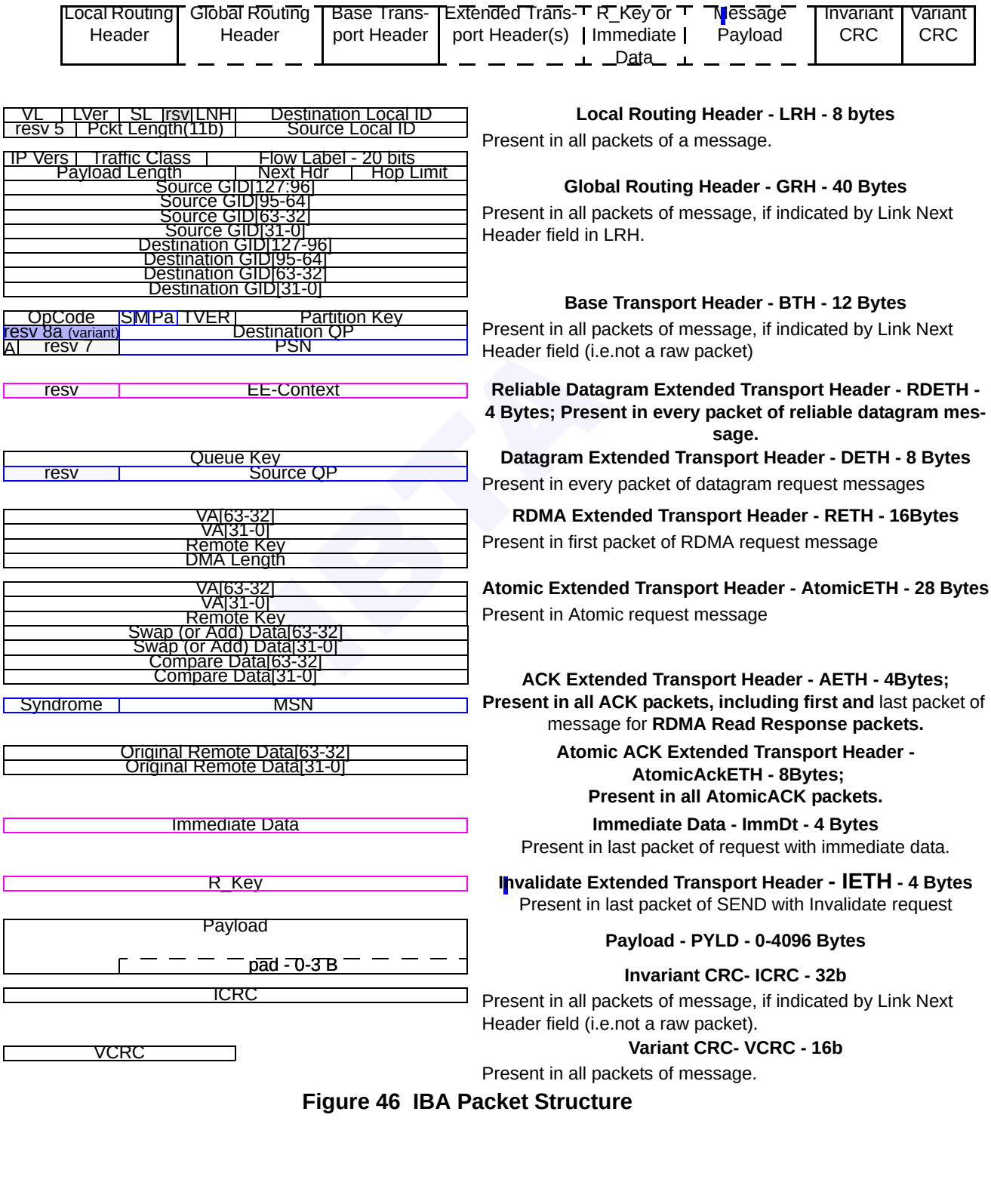


Figure 45 IBA Packet Overview

The IBA packet structure is shown in Figure 46 on page 153.



Local Routing Header - LRH - 8 bytes

Present in all packets of a message.

Global Routing Header - GRH - 40 Bytes

Present in all packets of message, if indicated by Link Next Header field in LRH.

Base Transport Header - BTH - 12 Bytes

Present in all packets of message, if indicated by Link Next Header field (i.e.not a raw packet)

Reliable Datagram Extended Transport Header - RDETH - 4 Bytes; Present in every packet of reliable datagram message.

Datagram Extended Transport Header - DETH - 8 Bytes

Present in every packet of datagram request messages

RDMA Extended Transport Header - RETH - 16Bytes

Present in first packet of RDMA request message

Atomic Extended Transport Header - AtomicETH - 28 Bytes

Present in Atomic request message

ACK Extended Transport Header - AETH - 4Bytes; Present in all ACK packets, including first and last packet of message for **RDMA Read Response packets.**

Atomic ACK Extended Transport Header - AtomicAckETH - 8Bytes; Present in all AtomicACK packets.

Immediate Data - ImmDt - 4 Bytes

Present in last packet of request with immediate data.

Invalidate Extended Transport Header - IETH - 4 Bytes

Present in last packet of SEND with Invalidate request

Payload - PYLD - 0-4096 Bytes

Invariant CRC- ICRC - 32b

Present in all packets of message, if indicated by Link Next Header field (i.e.not a raw packet).

Variant CRC- VCRC - 16b

Present in all packets of message.

Figure 46 IBA Packet Structure

5.2.1 LOCAL ROUTE HEADER (LRH) - 8 BYTES

C5-2: Packets generated by an InfiniBand device shall conform to the packet header format for the LRH as defined in table 4.

The Local Routing Header (LRH) contains fields used for local routing by switches within a IBA subnet. The following table summarizes the fields in the LRH.:

Table 4 Local Route Header Fields

Field Name	Field Abbreviation	Field Size (in bits)	Description
Virtual Lane	VL	4	This field identifies the virtual lane that the packet is using.
Link Version	LVer	4	This field identifies the Link level protocol of this packet. This version applies to the general packet structure including the LRH fields and the variant CRC
Service Level	SL	4	This field indicates what service level the packet is requesting within the subnet.
Reserved		2	Transmitted as 0, ignored on receive.
Link Next Header	LNH	2	This field identifies the headers that follow the LRH.
Destination Local ID	DLID	16	This field identifies the destination port and path (data sink) on the local subnet.
Reserved		5	Transmitted as 0, ignored on receive.
Packet Length	PktLen	11	This field identifies the size of the Packet in four-byte words. This field includes the first byte of LRH to the last byte before the variant CRC. See 7.7.8 Packet Length (PktLen) - 11 bits on page 194 for details on max and min values of PktLen
Source Local ID	SLID	16	This field identifies the source port (injection point) on the local subnet.

The LRH fields are fully defined in 7.7 Local Route Header on page 192.

5.2.2 GLOBAL ROUTE HEADER (GRH) - 40 BYTES

C5-3: Packets generated by InfiniBand devices shall conform to the packet header format for the GRH as defined in table 5.

Global Route Header (GRH) contains fields for routing the packet between subnets. The presence of the GRH is indicated by the Link Next Header (LNH) field in the LRH. The layout of the GRH is the same as the IPv6 Header defined in RFC 2460. Note, however, that IBA does not define a relationship between a device GID and IPv6 address (I.e. there is no defined mapping between GID and IPv6 address for any IB device or port).

The following table summarizes the fields in the GRH.

Table 5 Global Route Header Fields

Field Name	Field Abbreviation	Field Size (in bits)	Description
IP Version	IPVer	4	This field indicates version of the GRH
Traffic Class	TClass	8	This field is used by IBA to communicate global service level.
Flow Label	Flow-Label	20	This field identifies sequences of packets requiring special handling.
Payload length	PayLen	16	For an IBA packet this field specifies the number of bytes starting from the first byte after the GRH, up to and including the last byte of the ICRC. For a raw IPv6 datagram this field specifies the number of bytes starting from the first byte after the GRH, up to but not including either the VCRC or any padding, to achieve a multiple of 4 byte packet length. For raw IPv6 datagrams padding is determined from the lower 2 bits of this GRH:PayLen field. Note: GRH:PayLen is different from LRH:PkyLen.
Next Header	NxtHdr	8	This field identifies the header following the GRH. This field is included for compatibility with IPV6 headers. It should indicate IBA transport.
Hop Limit	HopLmt	8	This field sets a strict bound on the number of hops between subnets a packet can make before being discarded. This is enforced only by routers.
Source GID	SGID	128	This field identifies the Global Identifier (GID) for the port which injected the packet into the network.
Destination GID	DGID	128	This field identifies the GID for the port which will consume the packet from the network.

5.2.3 BASE TRANSPORT HEADER (BTH) - 12 BYTES

C5-4: Packets generated by an InfiniBand device shall conform to the packet header format for the BTH as defined in table 6.

Base Transport Header (BTH) contains the fields for IBA transports. The presence of BTH is indicated by the Next Header field of the last previous header (i.e either LRH:LNH or GRH:NextHdr depending on which was the

last previous header). The following table summarizes the fields in the BTH.:

Table 6 Base Transport Header Fields

Field Name	Field Abbreviation	Field Size (in bits)	Description
Opcode	OpCode	8	This field indicates the IBA packet type. The OpCode also specifies which extension headers follow the Base Transport Header
Solicited Event	SE	1	This bit indicates that an event should be generated by the responder.
MigReq	M	1	This bit is used to communicate migration state.
Pad Count	PadCnt	2	This field indicates how many extra bytes are added to the payload to align to a 4 byte boundary.
Transport Header Version	TVer	4	This field indicates the version of the IBA Transport Headers.
Partition Key	P_KEY	16	This field indicates which logical Partition is associated with this packet (see 10.9 Partitioning on page 523)
Reserved (variant)		8	Transmitted as 0, ignored on receive. This field is not included in the invariant CRC. see 7.8 CRCs on page 195 for details.
Destination QP	DestQP	24	This field indicates the Work Queue Pair Number (a.k.a. QP) at the destination
Acknowledge Request	A	1	This bit is used to indicate that an acknowledge (for this packet) should be scheduled by the responder.
Reserved		7	Transmitted as 0, ignored on receive. This field is included in the invariant CRC.
Packet Sequence Number	PSN	24	This field is used to detect a missing or duplicate Packet. See 9.7.1 Packet Sequence Numbers (PSN) on page 282 for a detailed description of PSN.

The detailed definition of the Base Transport Header fields are defined in Section 9.2 on page 234.

5.2.4 RELIABLE DATAGRAM EXTENDED TRANSPORT HEADER (RDETH) - 4 BYTES

o5-1: Packets generated by an InfiniBand device that supports reliable datagrams shall conform to the packet header format for the RDETH header as defined in table 7.

Reliable Datagram Extended Transport Header (RDETH) contains the additional transport fields for reliable datagram service. The RDETH is only

in Reliable Datagram packets as indicated by the Base Transport Header Opcode field. The following table summarizes the fields in the RDETH.:

Table 7 Reliable Datagram Extended Transport Header Fields

Field Name	Field Abbreviation	Field Size (in bits)	Description
Reserved		8	Transmitted as 0, ignored on receive.
EE-Context	EECnxt	24	This field indicates which End-to-End Context should be used for this Reliable Datagram packet

The detailed definition of the Reliable Datagram Extended Transport Header is in Section 9.3.1 Reliable Datagram Extended Transport Header (RDETH) - 4 Bytes on page 240.

5.2.5 DATAGRAM EXTENDED TRANSPORT HEADER (DETH) - 8 BYTES

C5-5: Packets generated by an InfiniBand device shall conform to the packet header format for the DETH as defined in table 8.

Datagram Extended Transport Header (DETH) contains the additional transport fields for datagram service. The DETH is only in datagram packets if indicated by the Base Transport Header Opcode field. The following table summarizes the fields in the DETH.:

Table 8 Datagram Extended Transport Header Fields

Field Name	Field Abbreviation	Field Size (in bits)	Description
Queue Key	Q_Key	32	This field is required to authorize access to the receive queue.
Reserved		8	Transmitted as 0, ignored on receive.
Source QP	SrcQP	24	This field indicates the Work Queue Pair Number (a.k.a. QP) at the source.

The detailed definition of the Datagram Extended Transport Header is in Section 9.3.2 Datagram Extended Transport Header (DETH) - 8 Bytes on page 240.

5.2.6 RDMA EXTENDED TRANSPORT HEADER (RETH) - 16 BYTES

o5-2: Packets generated by an InfiniBand device that supports RDMA operations shall conform to the packet header format for the RETH as defined in table 9.

RDMA Extended Transport Header (RETH) contains the additional transport fields for RDMA operations. The RETH is present in only the first (or only) packet of an RDMA Request as indicated by the Base Transport Header Opcode field. The following table summarizes the fields in the RETH.:

Table 9 RDMA Extended Transport Header Fields

Field Name	Field Abbreviation	Field Size (in bits)	Description
Virtual Address	VA	64	This field is the Virtual Address of the RDMA operation.
Remote Key	R_Key	32	This field is the Remote Key that authorizes access for the RDMA operation.
DMA Length	DMALen	32	This field indicates the length (in Bytes) of the DMA operation.

The detailed definition of the RDMA Extended Transport Header is in 9.3.3 RDMA Extended Transport Header (RETH) - 16 Bytes on page 241.

5.2.7 ATOMIC EXTENDED TRANSPORT HEADER (ATOMICETH) - 28 BYTES

o5-3: Packets generated by an InfiniBand device that supports atomic operations shall conform to the packet header format for the AtomicETH header as defined in Table 10.

Atomic Extended Transport Header (AtomicETH) contains the additional transport fields for Atomic packets. The AtomicETH is only in Atomic packets as indicated by the Base Transport Header Opcode field. The following table summarizes the fields in the AtomicETH.:

Table 10 Atomic Extended Transport Header Fields

Field Name	Field Abbreviation	Field Size (in bits)	Description
Virtual Address	VA	64	This field is the remote virtual address.
Remote Key	R_Key	32	This field is the Remote Key that authorizes access to the remote virtual address.
Swap (or Add) Data	SwapDt	64	This field is an operand in atomic operations.
Compare Data	CmpDt	64	This field is an operand in CmpSwap atomic operation.

The detailed definition of the Atomic Extended Transport Header is in Section 9.3.4 ATOMIC Extended Transport Header (AtomicETH) - 28 Bytes on page 242).

5.2.8 ACK EXTENDED TRANSPORT HEADER (AETH) - 4 BYTES

C5-6: Packets generated by an InfiniBand device shall conform to the packet header format for the AETH as defined in table 11.

ACK Extended Transport Header (AETH) contains the additional transport fields for ACK packets. The AETH is only in Acknowledge, RDMA READ Response First, RDMA READ Response Last, and RDMA READ Response Only packets as indicated by the Base Transport Header Opcode field. The following table summarizes the fields in the AETH.

Table 11 ACK Extended Transport Header Fields

Field Name	Field Abbreviation	Field Size (in bits)	Description
Syndrome	Syndrome	8	This field indicates if this is an ACK or NAK packet plus additional information about the ACK or NAK.
Message Sequence Number	MSN	24	This field indicates the sequence number of the last message completed at the responder.

The detailed definition of the ACK Extended Transport Header is in Section 9.3.5 on page 243.

5.2.9 ATOMIC ACK EXTENDED TRANSPORT HEADER (ATOMICACKETH) - 8 BYTES

o5-4: Packets generated by an InfiniBand device that supports atomic operations shall conform to the packet header format for the AtomicAckETH as defined in table 12.

Atomic ACK Extended Transport Header (AtomicAckETH) contains the additional transport fields for AtomicACK packets. The AtomicAckETH is only in Atomic Acknowledge packets as indicated by the Base Transport Header Opcode field. The following table summarizes the fields in the AtomicAckETH.:.

Table 12 Atomic ACK Extended Transport Header Fields

Field Name	Field Abbreviation	Field Size (in bits)	Description
Original Remote Data	Orig-RemDt	64	This field is the return operand in atomic operations and contains the data in the remote memory location before the atomic operation.

The detailed definition of the Atomic ACK Extended Transport Header is in Section 9.3.5.3 on page 243.

5.2.10 IMMEDIATE DATA EXTENDED TRANSPORT HEADER (IMMDT) - 4 BYTES

Immediate DataExtended Transport Header (ImmDt) contains the additional data that is placed in the receive Completion Queue Element (CQE). The ImmDt is only in Send or RDMA-Write packets with Immediate Data if indicated by the Base Transport Header Opcode.

The detailed definition of the Immediate Data Extended Transport Header is in Section 9.3.6 on page 244.

Note, the terms *Immediate Data Extended Transport Header* and *Immediate Data* are used synonymous in this specification.

5.2.11 INVALIDATE EXTENDED TRANSPORT HEADER (IETH) - 4 BYTES

The Invalidate Extended Transport Header (IETH) contains an R_Key field which is used by the responder to invalidate a memory region or memory window once it receives and executes the SEND with Invalidate request.

The detailed definition of the Invalidate Extended Transport Header is in Section 9.3.7 on page 244

5.2.12 PAYLOAD

Payload (PYLD) contains the application data being transferred end to end. Payload is not present in RDMA Read Requests, Acknowledge, CmpSwp, FetchAdd, and Atomic Acknowledge packets. It is optionally present in the other packet op-codes.

C5-7: The length of the Payload shall be 0 or more bytes up to the full path MTU.

C5-8: All packets of an IBA message that contain a payload shall fill the payload to the full path MTU except the last (or only) packet of the message.

C5-9: In a packet using InfiniBand transport, a Pad field of 0-3 bytes shall be included in the packet and used to align the Payload to a multiple of 4 bytes (i.e. the size of the Payload plus the Pad field is always a multiple of four bytes). The actual size of the Pad field used in a given packet shall be indicated in the Base Transport Header PadCnt field of the packet.

5.2.13 INVARIANT CRC

Invariant CRC (ICRC) covers the fields that do not change in packet from source to destination. ICRC is only in IBA packets, and is not present in

Raw Packets. Which fields are covered in the ICRC is dependent on the presence of the GRH.

The detailed definition of the Invariant CRC is in Section 7.8.1 on page 195.

5.2.14 VARIANT CRC

Variant CRC (VCRC) covers the fields that can change from link to link. The VCRC is in all packets, both IBA and Raw Packets. The VCRC can be regenerated in the fabric.

The detailed definition of the Variant CRC is in Section 7.8.2 on page 197.

5.3 RAW PACKET FORMAT

A Raw Packet is a packet that does not use IBA transport. Raw packets are not a required feature of InfiniBand devices, but if they are supported, the raw packet shall be formatted as specified in this section.

o5-5: If a Raw packet contains an IPv6 Routing Header, the packet structure shall be: LRH, IPv6, Payload (including any transport headers), and VCRC. If a Raw packet does not contain a IPv6 Routing Header, then the structure shall be: LRH, RWH, Payload, and VCRC.

o5-6: The RWH is a 32 bit “Raw Header” that shall contain the EtherType of the payload. EtherType indicates the protocol of the raw packet and shall conform to the definition in the IEEE Type Field Registrar. (See standards IEEE 802.3, 1998 Clause 3.2.6 Length/Type Field specifications and IEEE 802.1H-1995 for use of the Type Field.)

This format of “Raw” packets is shown in Figure 45 on page 152.

o5-7: The length of a raw packet (from after the RWH to before the variant CRC) must be a multiple of 4 bytes.

o5-8: The format of the Raw Header shall be as is shown in Figure 47.

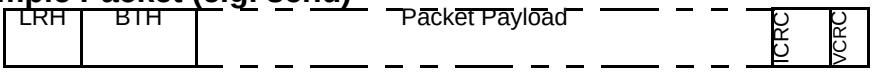
bits bytes	31-24	23-16	15-8	7-0
0-3	Reserved (Send as 0, ignore on receive)		EtherType	

Figure 47 Raw Header (RWH)

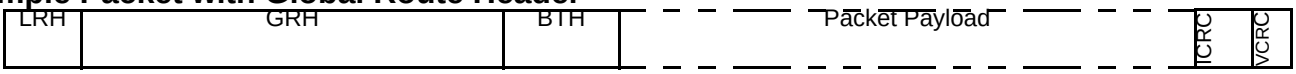
5.4 PACKET EXAMPLES

Some examples of IBA packets are shown in Figure 48.

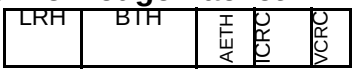
Simple Packet (e.g. send)



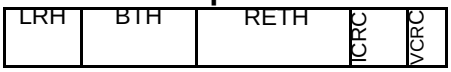
Simple Packet with Global Route Header



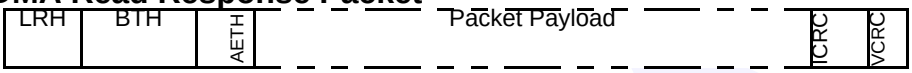
Acknowledge Packet



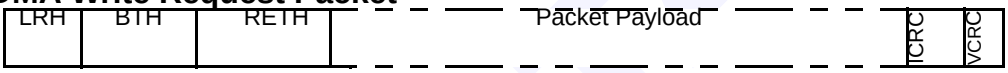
RDMA Read Request Packet



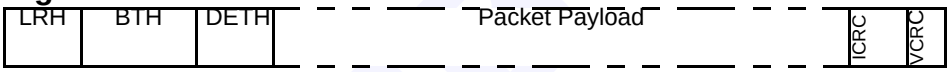
RDMA Read Response Packet



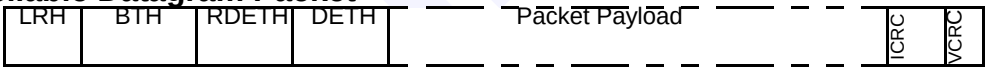
RDMA Write Request Packet



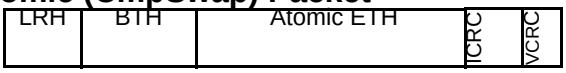
Datagram Packet



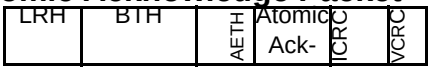
Reliable Datagram Packet



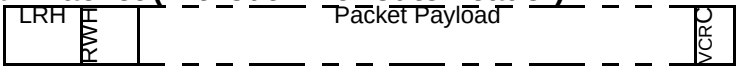
Atomic (CmpSwap) Packet



Atomic Acknowledge Packet



Raw Packet (without IPv6 route header)



Raw Packet (with IPv6 route header)

Figure 48 IBA Packet Examples

CHAPTER 6: PHYSICAL LAYER INTERFACE

6.1 OVERVIEW

This chapter describes services provided by the physical layer to the link layer and the logical interface between these layers. The physical layer also has an interface to management which is not covered in this chapter.

The description of the physical layer is provided in Volume 2, the electro-mechanical specification

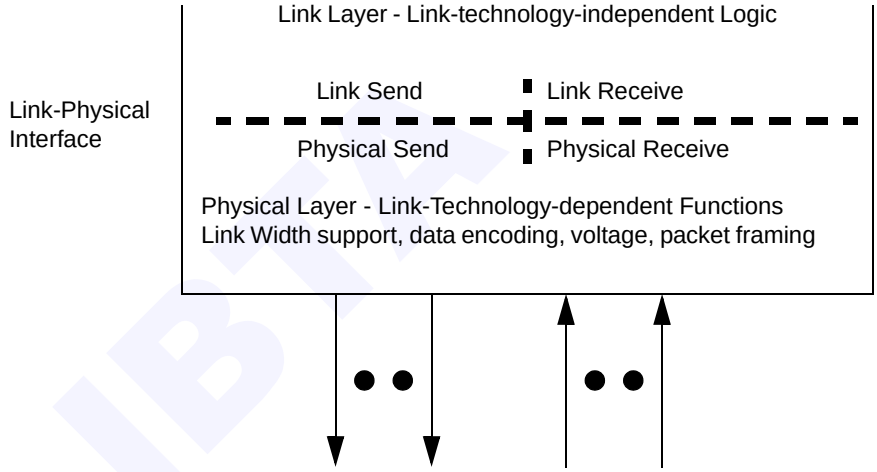


Figure 49 Physical Functions and Physical/Link Interface

6.2 SERVICES PROVIDED BY THE PHYSICAL LAYER.

The physical layer is responsible for:

- establishing a physical link when possible,
- informing the link layer whether the physical link is up or down,
- monitoring the status of the physical link, and
- when the physical link is up:
  - delivering received control and data bytes to the link layer, and
  - transmitting control and data bytes from the link layer.

See volume 2 for physical layer specifications.

6.3 INTERFACE BETWEEN PHYSICAL AND LINK LAYERS.

This chapter does not intend to describe an actual interface within a chip - it describes the functionality of the interface between the link-technology-dependent physical send and receive functions, and the link-technology-independent link logical function.

This interface is designed to keep the link and higher layer interface independent of physical layer implementation. The physical layer deals with all details that are dependent on the characteristics of operation over a particular physical layer such as line code.

The purpose of describing a logical interface and the related state machines is to partition functions to describe external behavior of IBA devices as simply and clearly as possible. Such descriptions are not intended to imply details of the internal implementation of devices. For instance, the interface described here does not imply the width of the internal link path which will be implementation dependent.

6.3.1 INTERFACE BETWEEN PHYSICAL RECEIVE AND LINK RECEIVE.

The following messages are sent between the physical receive function and the link logic.

6.3.1.1 PHY_LINK - PHYSICAL LINK STATUS

This message conveys the status of the physical link from the physical receive function to the link logic. This message is sent when physical link status changes and can take the following values:

down	the physical link is not operational. Sent when the link is in any non-operational status including no receive signal or retraining in progress
up	the physical link is trained and operational

These values report the status of the physical link as needed by the link logic. Any finer grain information needed by management (e.g. no_signal or retraining) will be obtained by management from the physical layer rather than passed through the link layer.

6.3.1.2 L_INIT_TRAIN - LINK INITIATE RETRAINING

This message is a request for retraining of the physical link. It is sent from the link logic to the physical receive function when the link logic has detected a need to retrain the link. See Section 7.12.2, “Error Recovery Procedures,” on page 221 for usage of this message.

6.3.1.3 RCV_STREAM - RECEIVE STREAM

This message conveys the control and data stream decoded by the receiver from the physical receive function to the link logic. This message is sent once for each data byte and once for each control signal received. The idle signaling of the physical link is treated as one control signal. This message can take the following values:

data	data and link packet contents
error	code violation
SDP	start data packet delimiter
SLP	start link packet delimiter
EGP	end good packet delimiter
EBP	end bad packet delimiter
idle	idle

6.3.2 INTERFACE BETWEEN PHYSICAL TRANSMIT AND LINK TRANSMIT.

The following messages are sent between the physical transmit function and the link logic.

6.3.2.1 XMIT_STREAM - TRANSMIT STREAM

This message conveys the control and data stream from the link logic to the physical layer. This message is sent once for each data byte and once for each control signal to be sent. The idle message causes the physical send function to send idles until a new message is received. This message can take the following values:

data	data and link packet contents
SDP	start data packet delimiter
SLP	start link packet delimiter
EGP	end good packet delimiter
EBP	end bad packet delimiter
idle	idle

6.3.2.2 XMIT_READY - PHYSICAL TRANSMITTER READY

This message is sent from the physical transmit function to the link transmitter to indicate whether the physical transmit function is ready to start transmitting a new packet. This provides physical layer dependent pacing back to the link layer since many physical layers have constraints that pre-

vent sending continuous packet traffic. This message can take the following values:

rdy	ready for packet initiation
wait	hold off packet initiation

CHAPTER 7: LINK LAYER

7.1 OVERVIEW

This chapter describes the behavior of the link and specifies the link level operations for devices attached to an IBA network. The link layer handles the sending and receiving of data across the links at the packet level. Services provided by the link layer include addressing, buffering, flow control, error detection and switching.

State machines are used in this specification to define the logical operation of the link layer as externally visible. They are not intended to define internal details of implementation. For instance, the packet receiver state machine operates on data received from the link layer as a stream of bytes though it is expected that many implementations of the link layer will process multiple bytes of the data stream in parallel.

7.1.1 STATE MACHINE CONVENTIONS

State machines are described to provide a clear description of the external behavior of the devices. Their description is not intended to imply the internal implementation of IBA devices. Actual implementations must take into account other considerations such as efficiency and suitability to the implementation technology.

The state machines in this chapter use the following conventions:

- Each state is represented by a box.
- The top section of the box contains the state name.
- The bottom section of the box contains the actions which occur in the state.
- Transition arrows indicate state transitions which will be made when the expression next to the arrow is satisfied.
- A transition arrow which does not originate in a state indicates a global transition. Such a transition will occur regardless of the current state. For instance, in Figure 50 on page 170, there is a global transition into the LinkDown state.
- If no exit condition for a state is satisfied, the machine remains in the current state.
- “Or” is represented by “+”.
- “And” is represented by “*”.

- The state diagrams represent the primary specification for the functions they depict. When a conflict exists between a state diagram and descriptive text, the state diagram takes precedence.

7.2 LINK STATES

C7-1: This compliance statement is obsolete and has been replaced by C7-1.1.1:.

C7-1.1.1: A port shall control its state and overall operation as specified in Figure 50 Link State Machine on page 170 and Section 7.2.7, "State Machine Terms," on page 169.

The states LinkInitialize and LinkArm are used by subnet management to configure devices on the subnet. Refer to 14.3.6 Port State Change on page 855 for additional information on how these states are used.

The link state machine is depicted in Figure 50. The following is a description of the states of this state machine.

7.2.1 LINKDOWN STATE

In the LinkDown state, the physical link is not up (that is, the physical layer is sending phy_link=down to the link layer) and the link layer is idle. In this state the link layer discards all packets presented to it for transmission.

7.2.2 LINKINITIALIZE STATE

In the LinkInitialize state, the physical link is up (that is, the physical layer is sending phy_link=up to the link layer) and the link layer can only receive and transmit subnet management packets (SMPs) and flow control link packets. While in this state, the link layer discards all other packets received or presented to it for transmission.

7.2.3 LINKARM STATE

In the LinkArm state, the physical link is up and the link layer can receive and transmit SMPs and flow control link packets. Additionally, the link layer can receive all other packets but discards all non-SMP data packets presented to it for transmission.

A switch port which is moved from LinkArm to LinkActive by a packet may also be the output port for that packet. The port will not be activated until the VCRC has been checked for the packet. One data packet should be able to pass through the network causing all armed ports on switches in its path to transition to active. Therefore, it is important that such a packet not be dropped when it is forwarded to a port that has not yet transitioned to active.

C7-1.a1: A switch shall ensure that a packet which causes its output port to transition from Armed to Active is not dropped by the port while in the Armed state. A switch port may enable transmission of data packets while in the Armed state.

7.2.4 LINKACTIVE STATE

In the LinkActive state, the physical link is up and the link layer can transmit and receive all packet types.

7.2.5 LINKACTDEFER STATE

The LinkActDefer state is entered from the LinkActive state when the physical layer indicates a failure in the link. If the error persists, the LinkDownTimeout expires and the port state transitions to LinkDown state. If the physical layer recovers prior to LinkDownTimeout expiration, the port state machine returns to the LinkActive state. While in the LinkActDefer state, the link layer will not transmit or receive packets. It may process packets already received as it would in the corresponding states. It will drop packets presented to it for transmission.

The purpose of this state is to allow for retraining of the physical link without requiring reinitialization of the link and higher layers.

7.2.6 MANAGEMENT STATE CHANGE COMMANDS

Management can send commands to attempt to alter the link state by sending a set request to the link port state in PortInfo. Only values of Down, Arm and Active are valid for such set requests. Commands to change state to Arm or Active are only valid when they appear as an exit term for the current state.

C7-2: Any management state change command with a value other than Down, Arm, or Active shall not result in a state change.

C7-3: A management state change command which is not valid in the current state shall not result in a state change.

For instance, Active is only valid when the current state is LinkArm. If the command is not valid for the current state, it will not cause a state change.

7.2.7 STATE MACHINE TERMS

Reset - An internal signal to reset the interface.

Remote_init - a link packet with the flow control initialize Op code (see 7.9.4 Flow Control Packet on page 210) has been received and has passed the checks of the link packet check state machine.

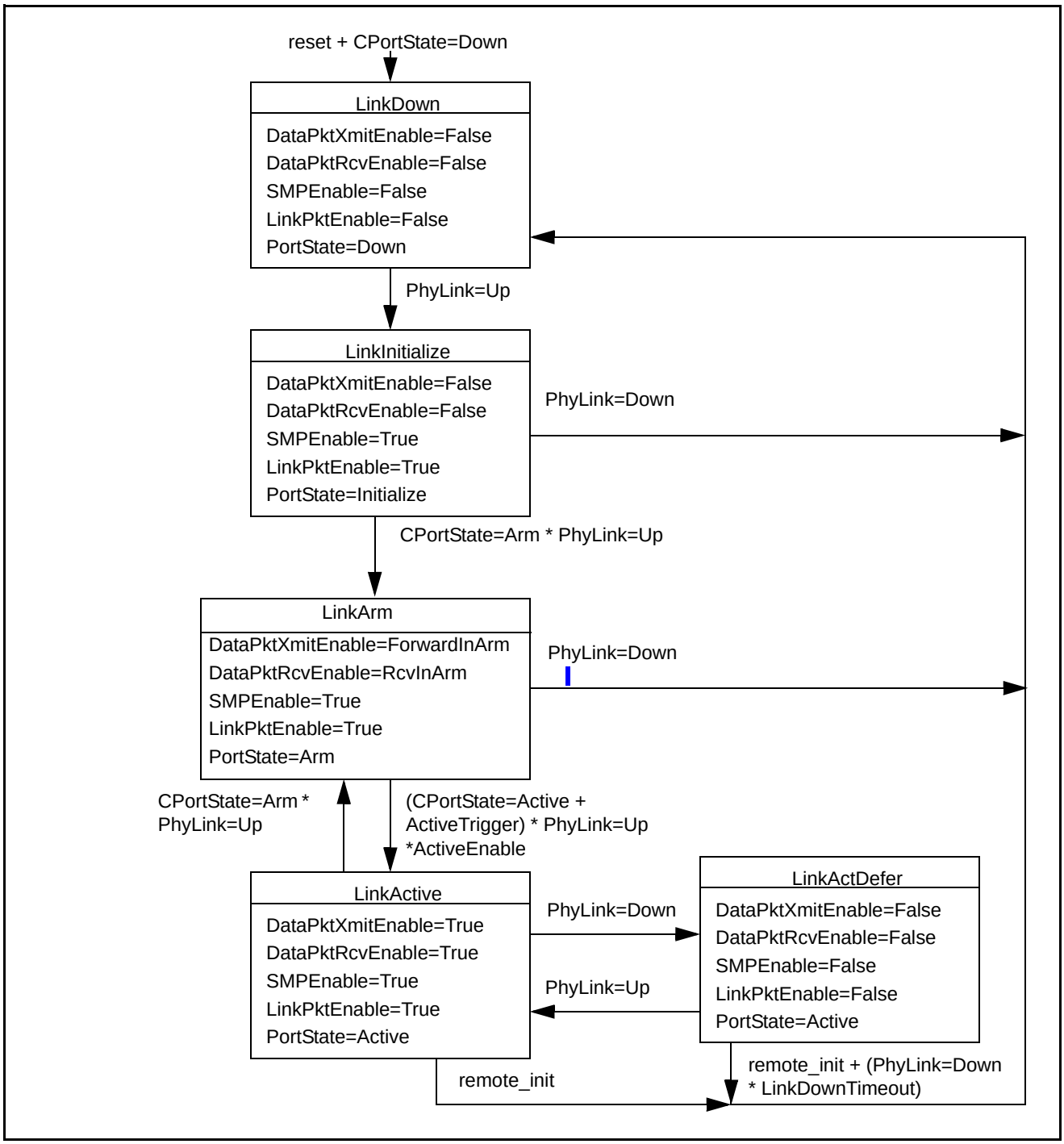


Figure 50 Link State Machine

Active_enable - a flag to prevent a premature transition from armed to active. It is set to false when the LinkInitialize state is exited. It is set to true when a link packet with the normal flow control Op code has been re-

ceived and has passed the checks of the link packet check state machine while in the LinkArm state.

PhyLink - the physical link status, phy_link, from the physical layer (refer to 6.3.1.1 Phy_link - Physical Link Status on page 164). Valid values are Up and Down.

PortState - the value of the PortState component of the PortInfo attribute. (Refer to 14.2.5.6 PortInfo on page 821.) Valid values are “Down”, “Initialize”, “Arm” and “Active”.

CPortState - a value that indicates commands from management to change the port state. Valid values are “Down”, “Arm”, and “Active”. Note that when phy_link=up and CPortState=down, the state machine will transition to the LinkDown state which will reset other link state machines. Since phy_link=up, this will be followed by a transition to the LinkInitialize state. Thus a command to change link port state to down provides a way to re-initialize the link layer. To disable a port requires a command to the physical layer port state machine. The value of CPortState shall only persist while in the state where it was received. If it satisfies a transition term from that state, it shall cause the transition. If it does not, it shall cause no transitions. Any state transition clears CPortState.

DataPktXmitEnable - a Boolean that indicates the link layer’s action with respect to transmission of non-SMP data packets. When True, transmission of non-SMP data packets is enabled. When False, non-SMP data packets submitted to link layer for transmission are discarded.

DataPktRcvEnable - a Boolean that indicates the link layer’s action with respect to reception of non-SMP data packets from the physical layer. When True, reception of non-SMP data packets is enabled. When False, non-SMP data packets received from the physical layer are discarded.

SMPEnable - a Boolean that indicates the link layer’s action with respect to transmission and reception of subnet management packets (SMPs). When True, transmission and reception of SMPs are enabled. When False, SMPs submitted to link layer for transmission or reception are discarded.

LinkPktEnable - a Boolean that indicates the link layer’s action with respect to transmission and reception of link packets. When True, transmission and reception of link packets are enabled. When False, link packets are not generated by the link layer and any link packets received are discarded.

ForwardInArm - a Boolean constant that indicates whether transmission of data packets is enabled during the Arm state. For a CA, this shall equal False. A switch may optionally use False or True.

RcvInArm - a Boolean constant that indicates whether data packets will be received during the Arm state. For a CA, this shall be equal to True. For a switch all ports will be set to True, except for ESP0 (if supported) which will be set to False (This is to guarantee that a virtual CA behind this ESP0 will not be able to inject non-VL15 packets into the fabric while the ESP0 is in the ARMED state. This is equivalent to the way a standalone CA would behave by means of its own port state while it is ARMED. Since virtual CAs behind ESP0 do not have their own port state, this mechanism that prevents injection of non-VL15 packets while ARMED, is based on the port state of the switch ESP0).

ActiveTrigger - a device dependent trigger that initiates the transition from LinkArm to LinkActive. For routers and channel adapters, ActiveTrigger occurs upon reception of a non-VL15 packet which passes the VCRC check on the port. For routers and channel adapters, ActiveTrigger is only generated on the port that received the packet.

For switches, ActiveTrigger occurs upon reception of a non-VL15 packet which passes the VCRC check on any port of the switch. Note, that for switches, the port receiving the packet could be in either Active or Armed state, and that ActiveTrigger is generated for all ports (including enhanced port 0) on the switch that are in Armed state.

LinkDownTimeout - a timeout that indicates that the physical link has been down (PhyLink = down) for a period of time that causes the port state machine to transition to the LinkDown state. LinkDownTimeout occurs when the port state machine has continuously been in the LinkActDefer state for 10ms +3% / -51%.

7.3 PACKET RECEIVER STATES

C7-4: This compliance statement is obsolete and has been replaced by C7-4.1.1:.

C7-4.1.1: Whenever the physical link is up, the packet receiver shall process the received stream from the physical layer as defined in Figure 51 Packet Receiver State Machine on page 174.

The packet receiver's primary input is the rcv_stream (refer to 6.3.1.3 rcv_stream - Receive Stream on page 165). See the Link/Phy Interface Chapter in Volume 2 of the IBA Specification for the definitions of rcv_stream=idle and data appearing this the state machine. (It should be noted that, for the Phy defined in IBA Version 1.1, and earlier, the rcv_stream=idle transition case from the *data pkt receive* and l*ink pkt receive* states cannot occur. This transition term is intended only for a Phy that provides explicit notification of *idle* at the receiver.)

The packet receiver monitors the received stream from the physical layer, rcv_stream, and passes any packets received with proper delimiters and no code violations to the link packet check or the data packet check as appropriate. Each byte of the rcv_stream is tested once by the state machine and causes at most one state transition. For example, when an SLP causes a transition from RcvDataPacket to BadPacket, that SLP does not cause a further transition from BadPacket to RcvLinkPacket.

While this logical state machine represents sending the whole packet to the packet checker once the end delimiter is received, implementations are allowed to begin processing the packet before that has occurred. Switches and routers may begin to forward a data packet while in the RcvDataPacket state if the packet passes all checks of the Data Packet Check state machine which require discard of the packet on failure. The required checks are all based on fields within the LRH. If further processing of the packet results in a transition to the MarkedBadPacket or BadPacket states and the switch or router has begun forwarding the packet, the switch or router shall corrupt the packet.

C7-5: To corrupt a packet, a switch or router shall place the 1's complement of the VCRC calculated for the transmitted packet in the VCRC field and shall terminate the packet with the EBP delimiter.

o7-1: When corrupting a packet, the switch or router may truncate the packet rather than sending all the received bytes.

C7-6: If a switch or router is forwarding a corrupted packet which is longer than indicated by the packet length field of the LRH, then it shall truncate the packet to less than or equal to the packet length field value.

C7-7: A CA shall not deliver a received packet to its client unless it has passed all the checks of the packet receiver and data packet check state machines.Therefore, when the action in the state is "discard or corrupt," a CA shall discard the packet.

Packets without proper start delimiters cause entry to the bad packet discard state and are discarded. Packets received with one or more bytes of rcv_stream=error or without proper end delimiters cause entry to the bad packet state and are discarded by CAs and discarded or corrupted by switches. The errors which cause entry to the bad packet discard and bad packet states indicate an error occurring on the local link. Packets received with no bytes of rcv_stream=error, a data packet start delimiter (SDP), and a bad packet end delimiter (EBP) indicate a packet forwarded by a switch that experienced an error that was not on the local link. These packets cause entry to the marked bad pkt state. Since link packets are not forwarded by switches and routers, they should never have a bad packet end delimiter. A packet with a start delimiter of SLP and an end de-

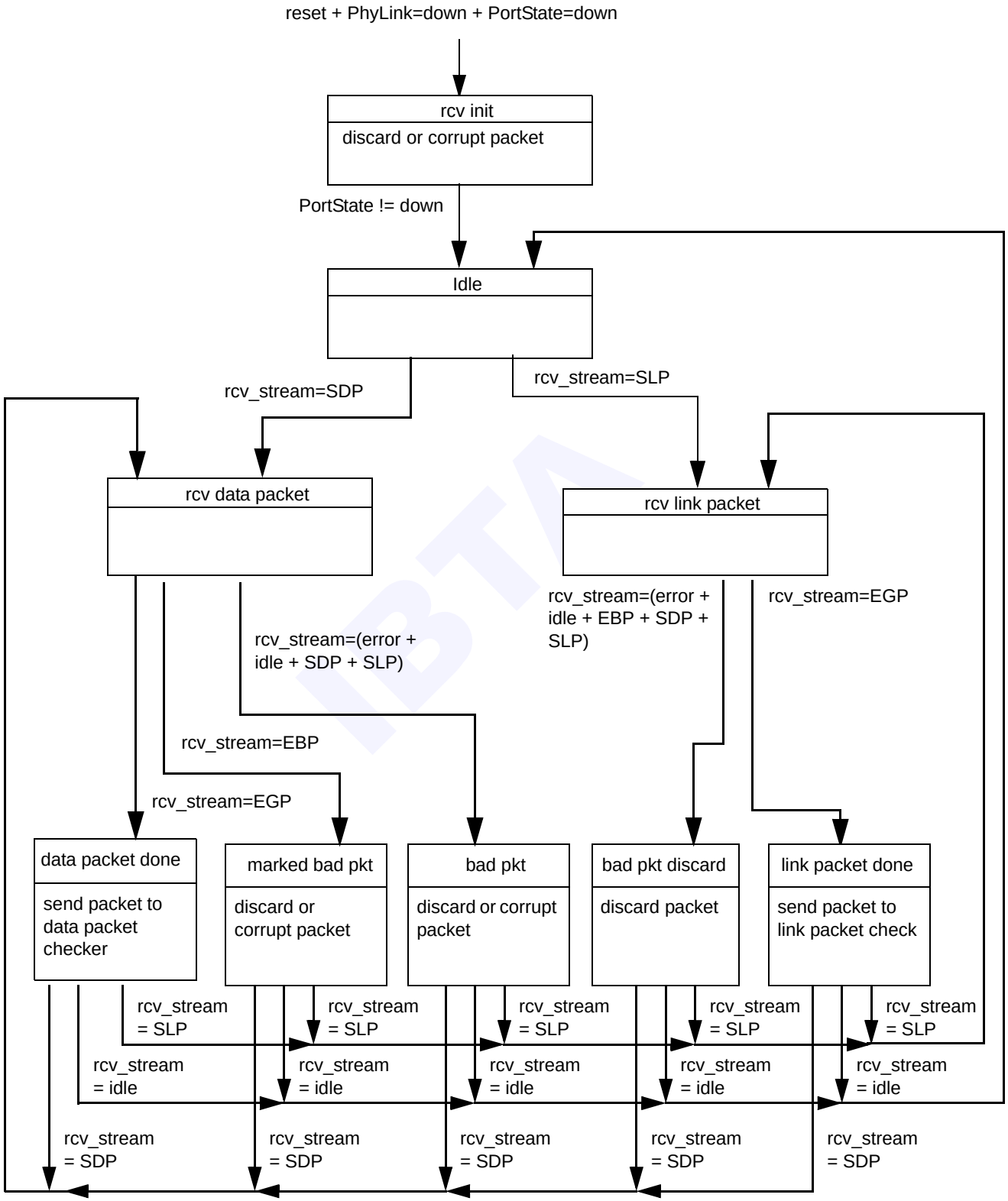


Figure 51 Packet Receiver State Machine

limiter of EBP is considered a local link error and causes entry to the bad packet state.

7.4 DATA PACKET CHECK

The data packet check machine in a CA verifies a data packet before passing it to the network layer. The data packet check machine in a switch or router port verifies a received data packet.

C7-8: This compliance statement is obsolete and has been replaced by C7-8.1.1:.

C7-8.1.1: Data packets shall be checked as specified by Figure 52 Data Packet Check machine on page 176 and Section 7.4, "Data Packet Check," on page 175. The order of checks within this state machine indicates the precedence of the errors for reporting and not necessarily the order in which the errors are detected.

For instance, most implementations would detect an invalid VL shortly after the packet starts and a CRC error cannot be detected until the end of the packet. However, CRC error is checked first in the state machine because if both of these errors occur, the CRC error indicates that the packet was damaged and that error should be reported rather than the VL error.

C7-9: A switch or router shall perform the same checks as a CA on packets for which the switch or router is the destination such as management packets addressed to the switch or router.

The data packet check machine in a CA passes packets to the receiver queueing. See Section 18.2.5.2 Receiver Queuing on page 1057.

C7-10: If a packet fails any test that terminates in a state of the Data Packet Check State Machine with the action "discard," switches, routers, and CAs shall discard the packet.

C7-11: For packets that only fail tests terminating in states of the Data Packet Check State Machine that specify the action of "corrupt or discard," a CA shall discard the packet and a switch or router shall discard the packet or corrupt it as defined in Section 7.3, "Packet Receiver States," on page 172.

VCRC_check

good	VCRC check was good
bad	otherwise

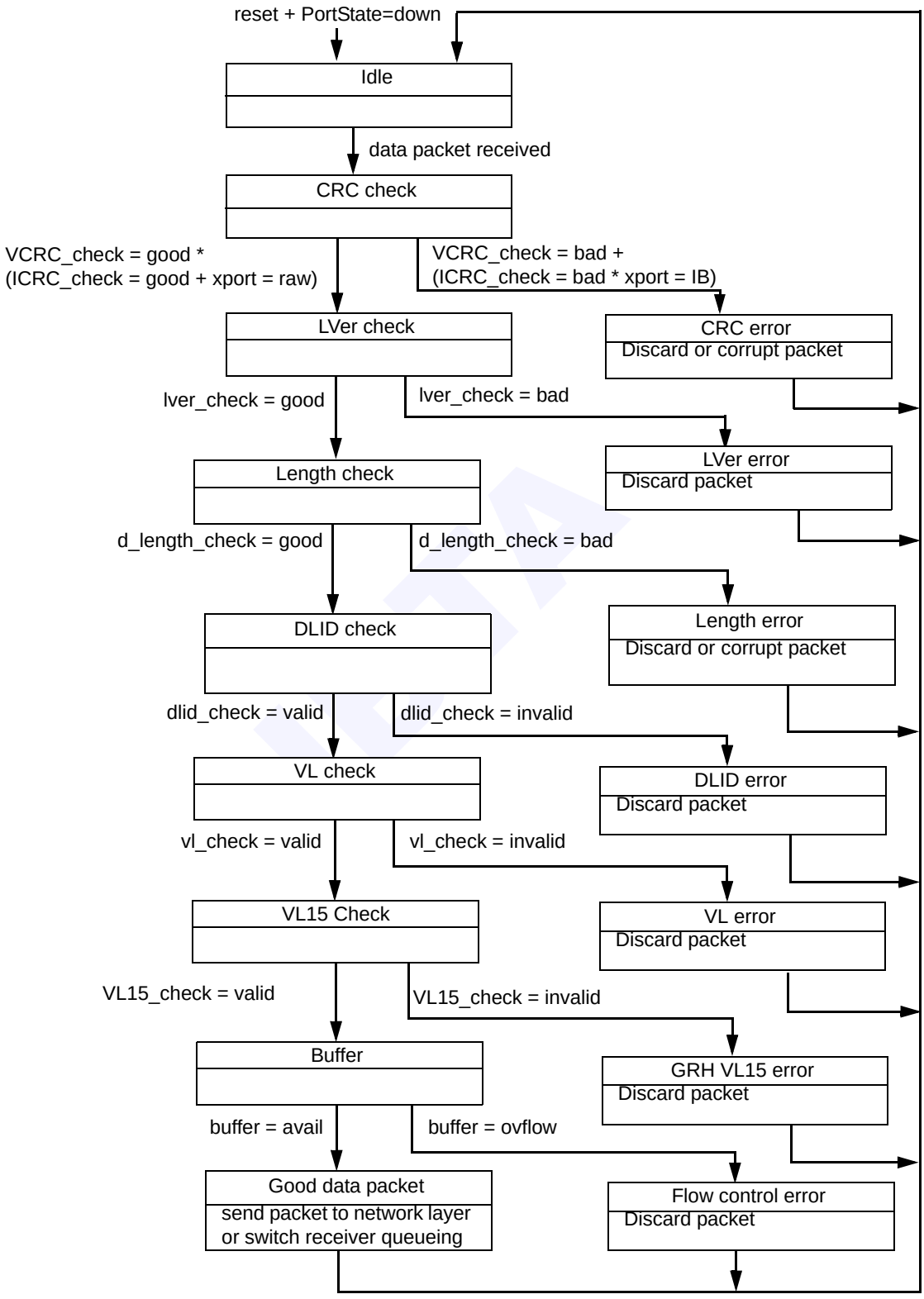


Figure 52 Data Packet Check machine

ICRC_check

good	ICRC check was good
bad	otherwise

The link layer of a switch or router is only required to check ICRC on packets that are destined to that switch or router. On all other packets, a switch or router may omit the ICRC check by returning ICRC_check = good without checking the ICRC.

xport

IB	LNH indicates IB transport
raw	LNH indicates raw transport

lver_check

good	LVer equals 0x0
bad	otherwise

d_length_check

good	PktLen * 4 = received data bytes - 2 and (MTU +124)/4 >= PktLen > = minimum length
bad	otherwise

Received length is the number of bytes between the SDP and EGP. MTU is PortInfo.MTUCap. Minimum length is 5 for raw packets and 6 for IBA transport packets. See Section 7.7.8, “Packet Length (PktLen) - 11 bits,” on page 194.

dlid_check

valid	for CAs: one of the following conditions is met: • DLID is a unicast LID of this CA Port, or • DLID is a multicast LID (i.e. in the range 0xc000 to 0xfffe), or • DLID is 0xFFFF (the permissive LID) and VL is 15 for switches and routers: DLID is not 0x0000.
invalid	otherwise

In addition to the above checks, if the DLID is a multicast LID a CA may optionally check if the DLID is configured for this CA. If the CA performs this check, the dlid_check result may be invalid if the DLID

is not configured for this CA. Thus a DLID which is within the range 0xC000 to 0xFFFE may be declared by the CA as being invalid if the specific DLID is not configured for this CA.

VL_check

valid	(VL is operational and PortState = Active or Armed) or (VL = 15 and DLID is unicast).[a]
invalid	otherwise

a. Note, the permissive LID is a unicast LID

VL15_check

valid	(VL <> 15) or (LNH indicates IBA local packet)
invalid	otherwise

buffer

avail	buffer is available for the packet
ovflow	otherwise

7.5 LINK PACKET CHECK

The only type of link packet currently defined is the flow control packet. See Section 7.9.4, “Flow Control Packet,” on page 210.

C7-12: A port shall verify a link packet as specified by Figure 53 Link Packet Check machine on page 179 and Section 7.5, “Link Packet Check,” on page 178 before passing it to the flow control.

LPCRC_check

good	LPCRC (Link Packet CRC) check was good
bad	otherwise

Op

flow	either flow control opcode is present
unknown	otherwise

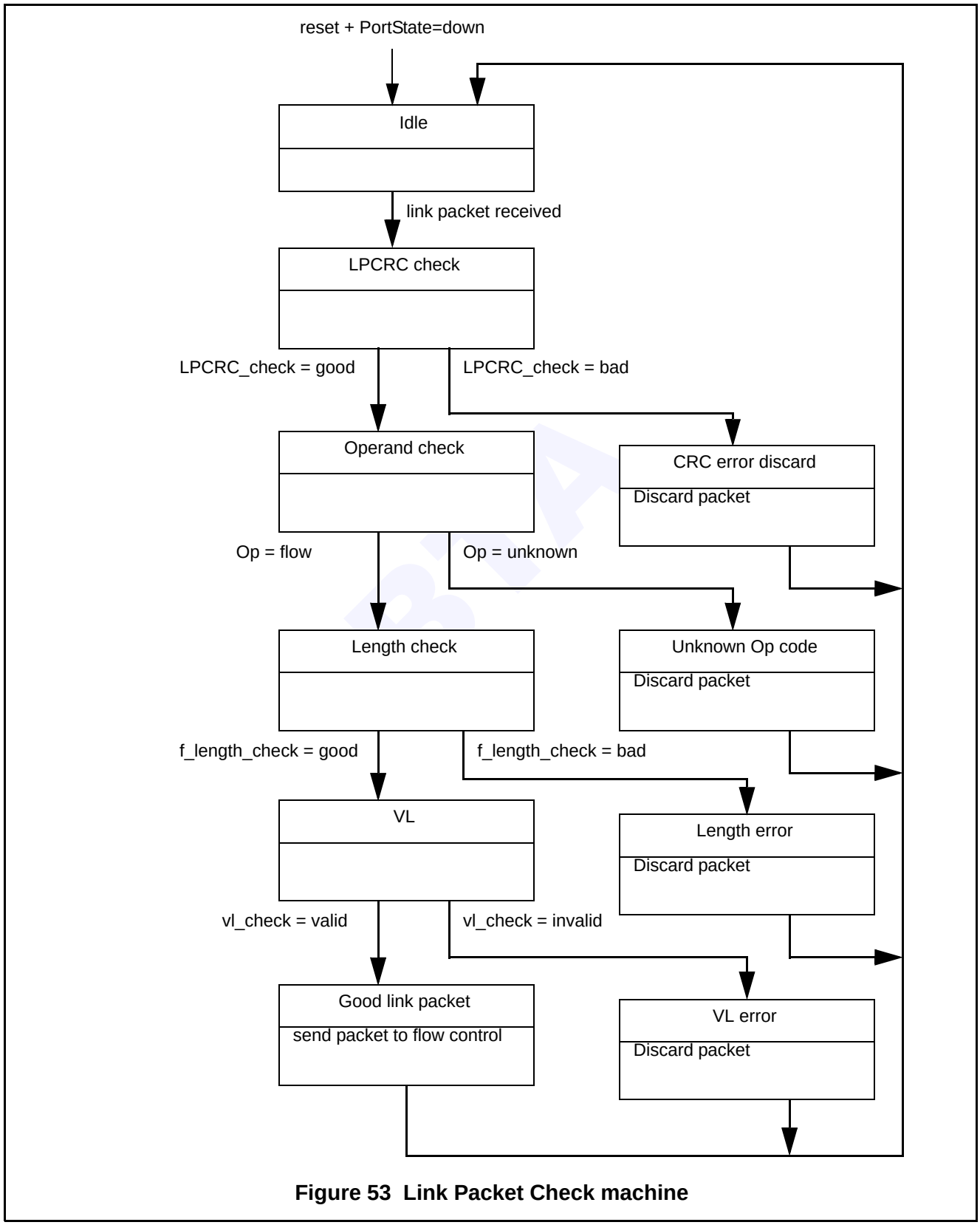


Figure 53 Link Packet Check machine

VL_check

valid	(VL is supported or PortState is not Active) and VL is not 15
invalid	otherwise

During initialization, the number of active VLs may not have been configured yet, so receiving credits for a non-supported VL is only an error when in the active state.

f_length_check

good	length received = 6 bytes (including LPCRC)
bad	otherwise

7.6 VIRTUAL LANES MECHANISMS

Virtual lanes (VLs) provide a means to implement multiple logical flows over a single physical link. Link level flow control can be applied to one lane without affecting the others. Table 13 on page 180 summarizes the key attributes of VLs.

C7-13: An InfiniBand protocol aware device shall conform to the requirements defined by the rows labeled *required VLs*, *buffering*, and *ordering* in Table 13.

o7-2: An InfiniBand protocol aware device that implements more than one data VL shall conform to the requirements defined by the row labeled *flow control* in Table 13.

Table 13 Key Virtual Lane Characteristics

Attribute	Description
VL	Represents a logical flow over a given physical link.
VL Types	There are two types of VLs, one for normal traffic called a data VL and one reserved for subnet management traffic. The subnet management traffic VL is VL15. All other VLs are for normal traffic.

Table 13 Key Virtual Lane Characteristics (Continued)

Attribute	Description
Required VLs	VL 15 shall be implemented in all IBA channel adapters, switches, and routers. VL 0 shall be implemented for application use in all IBA channel adapters, switches, and routers. VLs 1-14 may be implemented to support additional traffic segregation. If implemented, VLs shall be numbered as indicated in Table 14 VL Numbering and Interoperability on page 182
Buffering	Devices shall provide independent buffering resources for each VL. See 7.6.4 Buffering and Flow Control For Data VLs on page 183 for details.
Flow Control	Link-level flow control shall be implemented on a per VL basis. See 7.9 Flow Control on page 209 for description of flow control on data VLs. VL 15 does not use link-level flow control, however. See 7.6.3 Special VLs on page 182 for details. Flow control packets are not subject to flow control.
VL Field	4-bit field within the LRH indicating the actual VL being used by this packet.
SL Field	4-bit field located in the LRH indicating the requested service level within the local subnet. See 7.6.5 Service Level on page 185 for a description of this field.
Ordering	When fabric configuration is stable, unicast packets between the same source and destination LIDs within a subnet and using the same SL shall be ordered. Multicast packets shall also be similarly ordered. Note, however, that ordering is not guaranteed between unicast and multicast flows, even if on the same SL. Ordering is not maintained between different SLs. Packets on one SL may overtake packets on another SL, even if flowing through the same physical path within the fabric.

7.6.1 VL IDENTIFICATION

C7-14: The sending port of an InfiniBand protocol aware device shall identify each packet with the virtual lane to be used, this information being carried in the 4-bit VL field of link header. In addition, the local routing header shall contain a 4-bit Service Level (SL).

The use of the SL field is described in Section 7.6.5 on page 185.

7.6.2 NUMBER OF VLS SUPPORTED

C7-15: An InfiniBand protocol aware device shall conform to requirements defined by the rows labeled *VL numbering* and *configuration* in Table 14

C7-16: All ports of an InfiniBand protocol aware device shall support VL15. Further, all ports shall support data VL0.

o7-3: Ports may support more than one data VL. If they do, they shall do in accordance with the allowed number specified in Table 14 on page 182.

C7-17: The data VLs shall be numbered sequentially starting from zero.

Thus, if an implementation supports 4 data VLs, they shall be numbered 0, 1, 2 and 3.

Table 14 VL Numbering and Inter-operability

Number of Data VLs Supported	VL Numbering	List of VL Configurations That Shall Be Supported[a]
1	VL0	1
2	VL0, VL1	2, 1
4	VL0 - VL3	4, 2, 1
8	VL0 - VL7	8, 4, 2, 1
15	VL0 - VL14	15, 8, 4, 2, 1

a. Because the port at the other end of the link may support a different number of VLs, the port must support operation with different numbers of VLs.

7.6.3 SPECIAL VLS

VL 15 is a special VL and must be supported by all ports. The following lists the properties of VL 15:

C7-18: VL15 shall not be subject to flow control (both link level and end-to-end), i.e. VL 15 packets may be transmitted at any time.

C7-19: InfiniBand protocol aware devices shall discard VL15 packets if there is not enough room for reception. Other than the packet discard counter (16.1.3.5 PortCounters on page 945) this discard is done silently.

C7-20: All InfiniBand protocol aware devices shall support sourcing and sinking VL 15 packets.

C7-21: CAs and routers shall provide a minimum of a single packet buffer per port for VL15 on each port for reception.

C7-22: Switches shall provide a minimum of a single packet buffer for VL15 per switch.

C7-23: VL15 packets shall be scheduled preemptively, i.e. they are transmitted ahead of all other packets (including flow control packets).

C7-24: VL mapping in a switch does not apply to VL15. That is, a packet received by a switch on VL15 shall be transmitted on VL15 and no packet received on another VL shall be transmitted on VL15.

C7-25: This compliance statement is obsolete and has been removed.

The SL field should be set to 0 by devices sourcing VL15 packets and ignored by devices checking and sinking VL15 packets.

C7-26: VL15 packets shall not be forwarded between subnets, i.e. they shall not have a GRH and they shall not be raw.

C7-27: Packets using VL15 shall have a maximum payload of 256 payload bytes.

7.6.4 BUFFERING AND FLOW CONTROL FOR DATA VLS

Virtual Lanes provide independent data streams on the same physical link.

For data VLs, separate guaranteed buffering resources, and separate flow control shall be provided. (For VL 15, different flow control and buffering restrictions apply, and are described in above.)

C7-28: For data VLs, each VL on each port shall provide the appearance of separate buffering resources, i.e. although dedicated buffering resources are not required, the ports must behave as if they were.

C7-29: Each port shall advertise the number of credits available for each data VL configured using flow control packets.

These credit packets and the flow control process are described in 7.9 Flow Control on page 209.

Table 15 Processing of Link Packets on page 184 details the behavior of a port when sending and receiving a link packet for a given data VL. The

following terminology is used in this table (and elsewhere in this specification):

- A data VL is supported if its VL number is inside the range indicated by the PortInfo.VLCap attribute. This indicates that the data VL is supported by the port.
- A data VL is configured if its VL number is inside the range indicated by the PortInfo.OperationalVLs attribute.This indicates that the data VL is currently configured for use by the port.

(Refer to 14.2.5.6 PortInfo on page 821 for description of PortInfo.VLCap and PortInfo.OperationalVLs.)

C7-30: Each port shall send and receive link packets as specified in Table 15 Processing of Link Packets on page 184

Note, in this table, a required behavior has not been specified for the cases where the data VL is supported but not currently configured. This is done to support changing of the Data VL configuration. Note further, the Data VL configuration may be changed in any PortState including LInkActive. It should also be noted that changing Data VL configuration when in LinkActive state might result in generation of FlowControlUpdateErrors which in turn could cause transitions between LinkActive and LinkActive-Defer states and effect the values of the performance counter LinkErrorRecoveryCounter.

Table 15 Processing of Link Packets

PortState	Status of a Data VL	Sending of Credits on that Data VL	Receiving of Credits on that Data VL
LinkInitialize	Data VL is Configured	Shall send link packets for that Data VL	Shall be accepted
	Data VL is supported but not currently configured	May send link packets for that Data VL	Shall be ignored, no error
	Data VL is not supported	Shall not send link packets for that Data VL	Shall be ignored, no error
LinkArm or LinkActive	Data VL is Configured	Shall send link packets on that Data VL	Shall be accepted
	Data VL is supported but not currently configured	Should not send link packets on that Data VL	Shall be ignored, no error
	Data VL is not supported	Shall not send link packets on unsupported data VLs	Shall be discarded, malformed packet reported

C7-31: Each port shall provide sufficient buffering for each configured data VL to be able to advertise credit for at least one packet with MTU payload.

Note, MTU payload here refers to the lesser of MTUCap and neighborMTU for that port

(See 7.7.8 Packet Length (PktLen) - 11 bits on page 194 for definition of the corresponding packet size requirement.)

C7-32: When a data packet arrives at a port, it shall be placed in the buffer associated with that input port and VL field in the packet.

7.6.5 SERVICE LEVEL

Service Level (SL) is used to identify different flows within an IBA subnet. It is carried in the local route header of the packet.

C7-33: The SL of a packet shall not be changed as a packet crosses the subnet.

The SL is an indication as to the service class of the packet. IBA does not assign any specific meaning to an SL value. SLs are intended as a mechanism to aid in providing differentiated services, improved fabric utilization and avoiding deadlock. However, the specifics on how this is done is beyond the scope of this specification.

The IBA specification does, however, define two mechanisms using SLs and VLs that are intended as tools to implement Quality of Service (QoS) related services. One is SL-to-VL mapping, the other is data VL arbitration. Both are described in detail below.

o7-4: If multiple data VLs are supported, then both SL-to-VL mapping and data VL arbitration must be supported (both described below).

If only a single data VL is supported, then neither are required (although SL-to-VL mapping may still be implemented for SL filtering--see 7.6.6 VL Mapping Within a Subnet on page 186 for a description of this).

C7-34: The only requirement for devices supporting only a single data VL with respect to SLs and VLs is that the device shall include the SL value in the SL field when sourcing a packet into an IBA subnet.

Note that switches are included in this list because they can be the source of packets via their SMI or GSI interfaces. Note also that this specification does not require the validation of SL field at the packet destination.

There are no ordering guarantees between packets of different SL.

The source for SL for different transport services is detailed in 9.10 Header and Data Field Source on page 420. For connected services (unreliable connected, reliable connected and reliable datagrams), the SL associated with the forward and reverse paths of the same connection may be different (i.e. on the same connection, the SL associated with the DeviceA:transmitWQ may be different from that for the DeviceB:transmitWQ). For unreliable and raw datagrams, however, a node can always respond to a datagram from some other node using the same SL as the original datagram.

The SL used for a given destination (DLID), QOS, partition etc. is ultimately provided by the subnet manager. It may also be from derived sources such as request packets, local management agents etc.

7.6.6 VL MAPPING WITHIN A SUBNET

As a packet is routed across a subnet, it may be necessary for it to change VLs when it uses a given link. Examples of where this may be needed include:

1) The link may not support the VL previously used by the packet. This could happen when a device in the fabric supports a limited set of VLs.
2) Two traffic streams arriving on different input ports of a switch may be using the same outgoing link, and may also happen to be using the same VL when arriving at the switch. If VL mapping were not supported, then both traffic streams would have to use the same VL on the output port. VL mapping allows these two streams to be assigned different VLs on the outgoing links. In general, VL mapping offers greater flexibility in maintaining independent traffic flows within a fabric.

SL to VL mapping is used to change VLs as a packet crosses a subnet.

SL to VL mapping is required in channel adapters, switches, and routers that support more than one data VL. It is optional in those devices supporting only one data VL. If it is implemented it shall be implemented in accordance with the requirements of this section.

SL to VL mapping is done using a programmable mapping table. This is provided by the SLtoVLMappingTable.

o7-5: In channel adapters and routers that support SL to VL mapping, there shall be a logical table that maps the SL field in the packet LRH to the VL to be used for that output port. This table is 16 entries deep, with each port of the device having an independent table. All 16 possible values of SL shall be included in this table. The table indicates the VL number to be used by that packet when it is transmitted by the port.

o7-6: In switches that support SL to VL mapping, there shall be a logical table that maps the SL, input port and output port of the packet to the VL to be used for the next hop.

This table can be best viewed as a set of tables, one for each output port. Each of these per output port tables then indicates which VL should be used by the outgoing packet based on its SL field and the port that it arrived on. Because the switch supports an internal port (refer to 18.2.4.1 Switch Ports on page 1045) that will also source packets that require VL mapping, this port is included as one of the input ports in the table. See 14.2.5.8 SLtoVLMappingTable on page 835 for details on table size and layout.

o7-7: This compliance statement is obsolete and has been removed.

The table indicates the VL to be used by that packet for the next hop transmission based on packet SL, input port and output port.

This table provides mapping for the n+1 input ports (including the internal port) to n output ports.

Refer to Table 149 SLtoVLMappingTable on page 835 for details of on the SLtoVLMappingTable.

o7-8: Devices implementing SL to VL mapping shall behave as depicted in Table 16.

Table 16 SLtoVLMappingTable Behavior

VL Value in SLtoVLMappingTable	Action
VL15	Discard packet, no error.
Data VL not configured by port	Discard packet, no error.
Data VL configured by port	Forward packet to port using VL

The number of VLs supported is defined by the VLCap component of the PortInfo attribute, while the number configured is defined by the Operational VLs component of the PortInfo attribute. (Refer to 14.2.5.6 PortInfo on page 821) for description of PortInfo.VLCap and PortInfo.OperationalVLs.)

Note, the SLtoVLMappingTable may be programmed with VL15 for any SL that is not authorized to use that port (for channel adapters and routers) or input-output port path (for switches). As indicated by the above table, packets are discarded if the SLtoVLMappingTable returns VL15. This filtering is intended as a mechanism to help protect against unautho-

rized use of SLs, and to help in breaking routing dependency loops (and thereby avoiding routing deadlocks).

7.6.7 INITIALIZATION AND CONFIGURATION

In order to allow devices to be built with different numbers of VLs, the SM must be able to configure the number of VLs to be used on a given link. The SM can query each port to determine the number of VLs it supports and then configure to a number supported by both ports on the link. Table 14 on page 182 depicts the number of VLs combinations that each device must support. The number of VLs supported is defined by the PortInfo.VLCap component while the number of VLs configured is defined by the PortInfo.OperationalVL (Refer to 14.2.5.6 PortInfo on page 821).

Ports may be configured to 1, 2, 4, 8 or 15 VLs and must be configured to a value equal to or less than the number supported. If an attempt is made to program the OperationalVLs to a value larger than the VLCap, the port may load OperationalVLs with any valid value.

A port must be configured with the same number of VLs for both its sending and receiving directions.

Modification of the SLtoVLMappingTable may be made while the port is in operation.

o7-9: If a port implements SL-to-VL mapping, it shall not allow any packet in transit to be fragmented as a result of changing the SLtoVLMapping-Table contents.

Packets may be discarded or mis-mapped during this change, however.

When a channel adapter, router, or switch initializes, the SLtoVLMapping-Table is not required to be initialized (i.e.the contents are undefined). The table should be initialized by the SM prior to use by data traffic.

7.6.8 VL SCHEDULING AND FLOW CONTROL FOR VL15 AND FLOW CONTROL PACKETS

VL15 (i.e. subnet management packets) traffic and flow control packets will use preemptive scheduling. The order of precedence is depicted in Table 17.

7.6.9 VL ARBITRATION AND PRIORITIZATION

VL arbitration refers to the arbitration done for an outgoing link on a switch, router or channel adapter. Each output port has a separate arbiter. The arbiter selects the next packet to transmit from the set of candidate packets available for transmission on that port.

C7-35: The arbiter shall not violate packet ordering rules, i.e. packets on a given VL shall not be reordered.

The following describes the algorithm to be used by the VL arbiter.

7.6.9.1 VL ARBITRATION WHEN ONLY ONE DATA VL IS IMPLEMENTED

Table 17 depicts the arbitration rules for switch, router or channel adapters that implement only a single data VL. This is a simple priority scheme

Table 17 Arbitration Rules for Devices with only one data VL

Packet type	Precedence order
VL15	Highest
Flow control packet	2nd highest
VL0	Lowest

where all packets at a precedence level are sent before any packets at a lower precedence level.

o7-10: Devices implementing only a single data VL shall transmit packets on its output ports using the arbitration rules depicted in Table 17 Arbitration Rules for Devices with only one data VL on page 189.

7.6.9.2 VL ARBITRATION WHEN MULTIPLE DATA VL S ARE IMPLEMENTED

The implementation of multiple data VLs is an optional feature in IBA. If they are implemented, however, the implementation shall conform to the specification detailed in this section.

o7-11: For devices implementing more than one data VL, the transmission of VL15 packets and flow control packets shall be the same as depicted in Table 17 on page 189 except that here all the data VLs are at a lower priority than VL15 (highest) and flow control packets (second highest).

o7-12: Devices implementing more than one data VL shall also implement the algorithm described in Section 7.6.9.2 for arbitrating between packets on the data VLs.

A two level scheme is employed, using preemptive scheduling layered on top of a weighted fair scheme. Additionally, the scheme provides a method to ensure forward progress on the low-priority VLs. The weighting, prioritization, and minimum forward progress bandwidth is programmable.

VL arbitration is controlled by the VLArbitrationTable (refer to 14.2.5.9 VLArbitrationTable on page 836). This table shall consist of three components, *High-Priority*, *Low-Priority* and *Limit of High-Priority*. The *High-Priority* and *Low-Priority* components are each a list of VL/Weight pairs. The *High-Priority* list shall have a minimum length of one and a maximum of length of 64. The *Low-Priority* list shall have a minimum length equal to

the number of data VLs supported and a maximum of length of 64. The *High-Priority* and *Low-Priority* component lists are allowed to be of different length.

Each list entry shall contain (1) a VL number (values from 0-14), and (2) a weighting value (values 0-255), indicating the number of 64 byte units which may be transmitted from that VL when its turn in the arbitration occurs. The PktLen field in the LRH is used to determine the number of units in the packet. (Note, the VCRC and also the symbols between packets introduced by the physical layer should not be included in VL arbitration weight calculations.) The calculation shall be maintained to 4 byte increments.

A weight of 0 indicates that this entry should be skipped.

If a list entry is programmed for VL15 or for a VL that is not supported or is not currently configured by the port, the port may either skip that entry or send from any supported VL for that entry.

Note, that the same data VL may be listed multiple times in the High or Low-Priority component list, and, further, it can be listed in both lists.

Each configured data VL should be listed in at least one of the component lists. There is, however, no requirement for a device to check for this case. Should a configured data VL not appear in either component list, packets for this data VL may be dropped, sent when the arbiter has no packets to send or never sent.

The *Limit of High-Priority* component indicates the number of high-priority packets that can be transmitted without an opportunity to send a low priority packet. Specifically, the number of bytes that can be sent is *Limit of High-Priority* times 4K bytes, with the counting done the same as described above for weights (i.e. the calculation is done to 4 byte increments and a High-Priority packet can be sent if current byte count has not exceed exceeded the *Limit of High-Priority*). A value of 255 indicates that the byte limit is unbounded. (Note, it the 255 value is used, forward progress of low priority packets is not guaranteed by this arbitration scheme.) A value of 0 indicates that only a single packet from the high-priority table may be sent before an opportunity is given to the low-priority table.

The VLArbitrationTable may be modified when the port is active. This modification shall not result in fragmentation of any packet that is in transit. Arbitration rules may violated during this change, however.

When a channel adapter, router, or switch initializes, the VLArbitrationTable is not required to be initialized (i.e.the contents are undefined). The table should be initialized by the SM prior to use by data traffic.

7.6.9.2.1 ARBITRATION RULES BETWEEN VL15, LINK CONTROL AND DATA VL PACKETS

The rules of table Table 17 on page 189 apply, where the data VLs (VL0-VL14) have the lowest priority.

7.6.9.2.2 ARBITRATION RULES FOR DATA VL PACKETS

When there are no VL15 or Flow Control packets to send, the arbitration rules in this section apply.

7.6.9.2.3 ARBITRATION RULES BETWEEN HIGH AND LOW PRIORITY COMPONENTS

The *High-Priority* and *Low-Priority* components form a two level priority scheme. Each of these components (or tables) may have a packet available for transmission. A packet is available for transmission from the High Priority table if the following test succeeds:

For each entry in the High Priority table, determine if:

1) the VL field matches that of any packets that are currently waiting for transmission for this port AND
3) there is available credit to send that packet

An entry with 0 weight is considered not in the list.

Note, Implementations may check if HiPriAvailWeight is available in determining if a packet is available.

Upon completion of transmission of a packet the following test should be done to determine which table to use to transmit the next packet:

If the High-Priority table has an available packet for transmission (as defined above) and the HighPriCounter has not expired, then the High-Priority is said to be *active* and a packet may be sent from the High-Priority table.

If the High-Priority table does not have an available packet for transmission (as defined above), or if the HighPriCounter has expired, then the HighPriCounter shall be reset, the Low-Priority table is said to be *active* and a packet may be sent from the Low-Priority table.

The following rules govern the operation of the HighPriCounter:

1) The HighPriCounter expires when its current value is negative.
2) If the value in the *Limit of High-Priority* component is not 255, then for each High-Priority packet transmitted, the size of the packet (as defined by the PktLen field in the LRH) is deducted from the current value of the HighPriCounter. The calculation should be maintained to 4 byte increments.
3) When the HighPriCounter is reset, the value in the *Limit of High-Priority* component times 4K bytes is loaded into the HighPriCounter.

7.6.9.2.4 ARBITRATION RULES WITHIN THE HIGH AND LOW COMPONENTS

Within each High or Low Priority table, weighted fair arbitration is used, with the order of entries in each table specifying the order of VL scheduling, and the weighting value specifying the amount of bandwidth allocated to that entry. Each entry in the table is processed in order.

A separate pointer and available weight count is maintained for each of the two tables. The pointers identify the current entry in the table, while the available weight count indicates the amount of weight that the current entry has available for data packet transmission. When a table is active (as defined in the previous section), the current entry in the table is inspected. A packet corresponding to this entry will be sent to the output port for transmission and the packet size (in 4 byte increments) will be deducted from the available weight count for the current entry, if all of the following are true:

1) The available weight for the list entry is positive.
2) There is a packet available for the VL of the entry
3) Buffer credit is available for this packet.

Note, if the available weight at the start of a new packet is positive, condition 1 above is satisfied, even if the packet is larger than the available weight.

When any of these conditions is not true, the next entry in the table is inspected. The current pointer is moved to the next entry in the table, the available weight count is set to the weighting value for this new entry, and the above test repeated. This is repeated until a packet is found that can be sent to the port for transmission. If the entire table is checked and no entry can be found satisfying the above criteria, the other table becomes active.

This description depicts the logical flow of the arbitration process, but does not specify performance requirements. Implementations shall perform in a logically consistent manner with the above description. Implementations may process steps in parallel and may pipeline tests. As an example of pipelining of tests, the check that there be available packets may return false if a packet has just recently been forwarded to output port but the arbiter logic has not processed its arrival.

Further, implementations are not required to implement the pointers, available weight counter and HighPriCounter. They must, however, behave in a manner equivalent to that described in this section.

7.7 LOCAL ROUTE HEADER

Local Routing Header - LRH - 8 bytes

The Local Routing Header (LRH) contains the fields for local routing by switches within a IBA subnet. The LRH is at the start of every packet and the packet ends with the Variant CRC. The LRH is 8 bytes long. For additional information on overall packet layout, see Chapter 5: Data Packet Format on page 150.

bits bytes	31-24		23-16			15-8	7-0
0-3	VL	LVer	SL	Rsv2	LNH	Destination Local Identifier	
4-7	Reserve 5	Packet Length (11 bits)				Source Local Identifier	

Figure 54 Local Route Header (LRH)

C7-36: The LRH shall use the format specified in Figure 54 Local Route Header (LRH) on page 193.

7.7.1 VIRTUAL LANE (VL) - 4 BITS

Specifies a virtual lane to be used for a packet. This field identifies which receive buffer and which receive flow control credits should be used for the received packet.

C7-37: The VL field shall be set to the VL on which the packet is sent.

The virtual lane can change from link to link in a subnet. Since the Virtual Lane can change, the Link Virtual Lane is not included in the Invariant CRC field.

7.7.2 LINK VERSION (LVER) - 4 BITS

Specifies the version of the Local Routing Header used for this packet. This version applies to the general packet structure including the LRH fields and the variant CRC.

C7-38: The LVer field shall be set to 0x0.

If a receiving device does not support the Link Version specified then the packet is discarded.

7.7.3 SERVICE LEVEL (SL) - 4 BITS

The Service Level field. This field is used by switches to determine the Virtual Lane used for this packet. This is described in Section 7.6.5 on page 185.

7.7.4 RESERVE - 2 BITS

C7-39: The 2-bit Reserve field shall be transmitted as 00 and shall be ignored on receive.

7.7.5 LINK NEXT HEADER (LNH) - 2 BITS

Specifies what headers following the Local Routing Header. The first bit (msb) indicates whether the packet uses IBA transport. The second bit (lsb) indicates whether a GRH/IPv6 header is present.

Table 18 Link Next Header Definition

Packet Type	LNH bit 1 IBA Transport	LNH bit 0 GRH/IPv6 header	Transport	Next Header
IBA global	1	1	IBA	GRH
IBA local	1	0	IBA	BTH
IP - non-IBA transport	0	1	Raw	IPv6
Raw	0	0	Raw	RWH (Ethertype)

C7-40: The LNH field shall indicate the packet type of the following packet as defined by Table 18 Link Next Header Definition on page 194.

7.7.6 DESTINATION LOCAL IDENTIFIER (DLID) - 16 BITS

Specifies the LID of the port to which the subnet delivers the packet. LIDs are unique within a subnet. More specifically it identifies the route to take to the destination port. If the packet is to be routed to another subnet, then this is the LID of the Router.

7.7.7 RESERVE - 5 BITS

C7-41: The 5 bit reserve field shall be transmitted as 00000 and shall be ignored on receive.

7.7.8 PACKET LENGTH (PKTLEN) - 11 BITS

The number of 4 byte words contained in the packet.

C7-42: The value of the PktLen field shall equal the number of bytes in all the fields starting with the first byte of the Local Route Header and the last byte before the Variant CRC, inclusive, divided by 4.

The maximum allowable size of all headers plus the CRC fields is 126 bytes. The maximum value of this field is (4096 + 126 - 2)/4 = 4220 / 4= 1055, reflecting a maximum of 126 bytes for all headers and CRCs minus the uncounted variant CRC.Note, the current version of the IBA is limited to 106 bytes (I.e. The longest current packet header that is possible is an

RD Atomic Op (LRH + GRH + BTH + RDETH + DETH + AtomicETH + VCRC + ICRC). The additional allowable size is for future expansion.

Table 19 Packet Size

MTU	Maximum Packet Length (Bytes/4)	Maximum Bytes (MTU+126)
256	95	382
512	159	638
1024	287	1150
2048	543	2174
4096	1055	4222

C7-43: For packets with IBA transport, the smallest allowed value for Packet Length is 6 (24 Bytes) including LRH.

C7-44: For raw packets, the smallest allowed value for Packet Length is 5 (20 Bytes) including LRH.

C7-45: The maximum allowed value for Packet Length is the value shown in Table 19 Packet Size on page 195 for the smaller of MTUCap and NeighborMTU.

Note, this compliance statement defines the maximum size packet. The d_length_check specified in Section 7.4: Data Packet Check is done against PortInfo:MTUCap only, however.

7.7.9 SOURCE LOCAL IDENTIFIER (SLID) - 16 BITS

C7-46: For all non-directed route packets, the SLID shall be a LID of the port which injected the packet onto the subnet.

For requirements on DLID in directed route packets, see 14.2 Subnet Management Class on page 794.

The subnet manager assigns each node a LID which is unique within a subnet.

7.8 CRCs

7.8.1 INVARIANT CRC (ICRC) - 4 BYTES

Specifies a Cyclic Redundancy Code covering all the fields of the Packet which are invariant from end to end through all switches and routers on the network. This field is present in all IBA packets but is NOT present in Raw Packets because for raw packets it is not known which fields will be invariant. The CRC calculation is re-started with each packet in the message. Which header fields that are included depends on whether the

Global Routing Header is present because the router may modify additional header fields.

C7-47: The ICRC field shall be present in all IBA transport packets.

C7-48: The ICRC field shall be calculated as specified in Section 7.8.1, "Invariant CRC (ICRC) - 4 Bytes," on page 195.

If the packet is local to the subnet (the Global Routing Header is not present), then the ICRC calculation is as follows:

- With no GRH, the ICRC includes:
  - Local Routing Header: except for the VL.
  - Base Transport Header: except for the Resv8a field
  - Extension Transport Headers (if present),
  - Packet Payload (if present),
- With no GRH, the ICRC excludes: (these fields are replaced with 1s for the ICRC calculation)
  - Local Routing Header: VL,
  - Base Transport Header: Resv8a.

If the packet is routed between subnets, so the Global route header is present, the ICRC calculation is as follows:

- With a GRH, the ICRC includes:
  - Global Routing Header: Version, Payload length, Next Header, Source IPV6 address, and Destination IPV6 address
  - Base Transport Header, except for the Resv8a field,
  - Extension Transport Headers (if present),
  - Packet Payload (if present).
- With a GRH, the ICRC excludes: (these fields are replaced with 1's for the CRC calculation)
  - Local Routing Header, all fields,
  - Global Routing Header: Flow label, Traffic Class, and Hop Limit fields.
  - Base Transport Header: Resv8a.

All fields in the packet. including those excluded from the Invariant CRC, are protected by the Variant CRC described in the next section.

The polynomial used is the same CRC-32 used by Ethernet - 0x04C11DB7. The procedure for the calculation is:

1) The initial value of the CRC-32 calculation is 0xFFFFFFFF.
2) The CRC calculation is done in big endian byte order with the least significant bit of the most significant byte being the first bits in the CRC calculation.
3) The bit sequence from the calculation is complemented and the result is the ICRC.
4) The resulting bits are sent in order from the bit representing the coefficient of the highest term of the remainder polynomial. The least significant bit, most significant byte first ordering of the packet does not apply to the ICRC field.

The CRC always starts with LRH:LVer bit 0, whether GRH is present or not.

This bit and byte ordering is consistent with Ethernet's CRC calculation.

bits bytes	31-24	23-16	15-8	7-0
	Bit0 in CRC Calculation, Bit 0, Byte 0 ↓			
0-3	Byte0	Byte 1	Byte 2	Byte3
4-7	Byte 4	Byte 5	Byte 6	Byte 7
8-11	Byte 8	Byte 9	Byte 10	Byte 11
...				

Figure 55 CRC Calculation Order

7.8.2 VARIANT CRC (VCRC) - 2 BYTES

Specifies a Cyclic Redundancy Code covering all fields of the Packet. This field is present in all data packets including Raw Packets and includes all bytes from the first byte of the LRH to the last byte before the Variant CRC, inclusive. Since a number of these fields can change as the packet is processed by switches and routers the Variant CRC may have to regenerated at each Link through the subnet. If a switch does not change any fields including the Link Virtual Lane, then the Variant CRC does not have to be regenerated.

C7-49: The VCRC field shall be present in all data packets.

C7-50: The VCRC field shall be calculated as specified in Section 7.8.2, "Variant CRC (VCRC) - 2 Bytes," on page 197.

The polynomial used is the same CRC-16 used by HIPPI-6400 - 0x100B. The procedure for the calculation is:

1) The initial value of the CRC-16 calculation is 0xFFFF.

2) The CRC calculation is done in big endian byte order with the least significant bit of the first byte of the Local Route Header (bit 0 of LRH:LVer) being the first bit in the CRC calculation.
3) The bit sequence from the calculation is complemented and the result is the VCRC.
4) The resulting bits are sent in order from the bit representing the coefficient of the highest term of the remainder polynomial. The least significant bit, most significant byte first ordering of the packet does not apply to the VCRC field.

This bit and byte ordering is consistent with Ethernet's CRC calculation.

7.8.3 LINK PACKET CRC (LPCRC) - 2 BYTES

Specifies a Cyclic Redundancy Code covering all fields of the Link Packet. This field is present in all Link packets including Flow Control Link Packets and includes all bytes from the first byte of the Opcode to the last byte before the LPCRC, inclusive. This field is always computed for each Link-packet.

C7-51: The LPCRC field shall be present in all link packets.

C7-52: The LPCRC field shall be calculated as specified in Section 7.8.3, "Link Packet CRC (LPCRC) - 2 Bytes," on page 198.

The polynomial used is the same CRC-16 used by Variant CRC and HIPPI-6400 - 0x100B. The procedure for the calculation is:

1) The initial value of the CRC-16 calculation is 0xFFFF.
2) The CRC calculation is done in big endian byte order with the least significant bit of the first byte of the Local Route Header (bit 0 of LRH:LVer) being the first bit in the CRC calculation.
3) The bit sequence from the calculation is complemented and the result is the LPCRC.;
4) The resulting bits are sent in order from the bit representing the coefficient of the highest term of the remainder polynomial. The least significant bit, most significant byte first ordering of the packet does not apply to the LPCRC field.

This bit and byte ordering is consistent with Ethernet's CRC calculation.

7.8.4 CRC CALCULATION SAMPLES

The following is an example of CRC calculation. The requirements for the CRC calculation are specified above, this section is intended for informative purposes only.

7.8.4.1 ICRC GENERATOR

The polynomial used for ICRC calculation is 0x04C11DB7. The seed value is 0xFFFFFFFF. The ICRC Generator Remainder is the complement of the resulting calculation.

The ICRC Generator actual implementation is not specified. The diagram in Figure 56 is provided as a reference with the sole purpose of clarifying the calculation details and does not imply a required implementation.

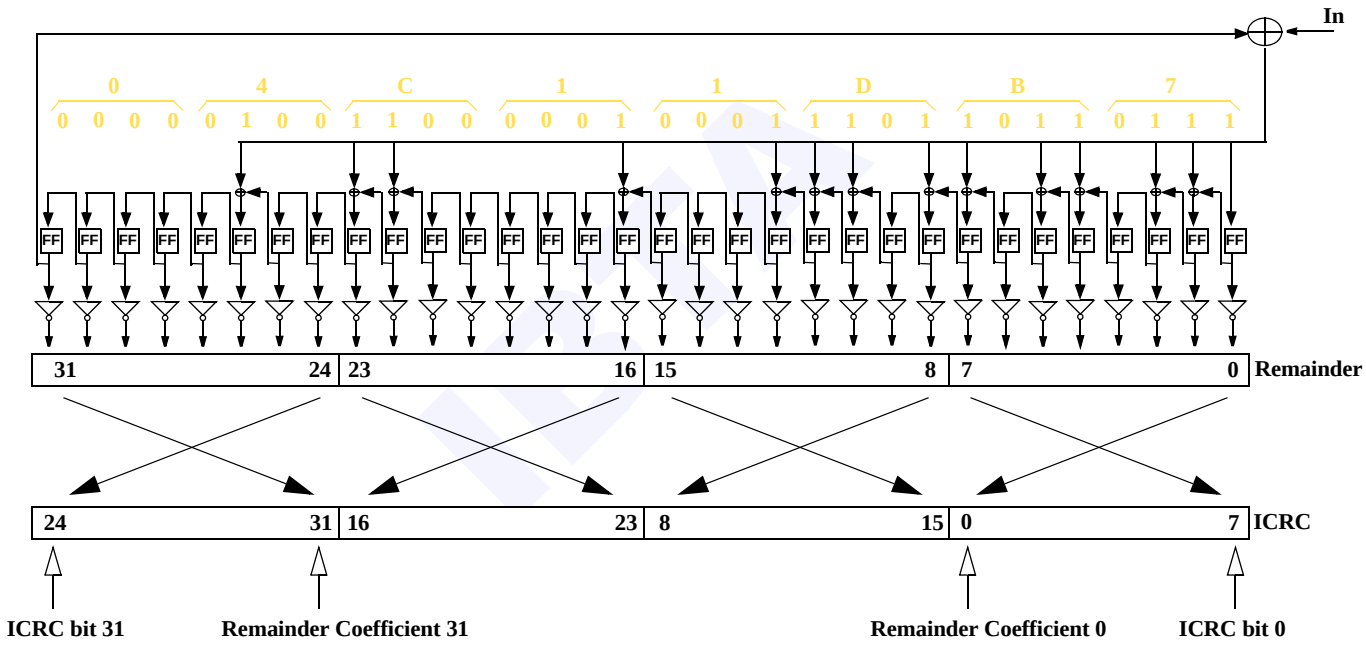


Figure 56 ICRC Generator

The 32 Flip-Flops are initialized to 1 before every ICRC generation. The packet is fed to the reference design of Figure 56 one bit at a time in the order specified in Section 7.8.1 on page 195. The **Remainder** is the ***bit-wise NOT*** of the value stored at the 32 Flip-Flops after the last bit of the packet was clocked into the ICRC Generator. **ICRC Field** is obtained from the **Remainder** as shown in Figure 56. **ICRC Field** is transmitted using Big Endian byte ordering like every field of an InfiniBand packet.

7.8.4.2 VCRC GENERATOR

The polynomial used for VCRC and FCCRC calculation is 0x100B. The seed value is 0xFFFF. The VCRC/FCCRC Generator Remainder is the complement of the resulting calculation.

The VCRC and FCCRC are generated in the same manner as described above for the ICRC. Figure 57 shows the reference design for the VCRC / FCCRC Generator.

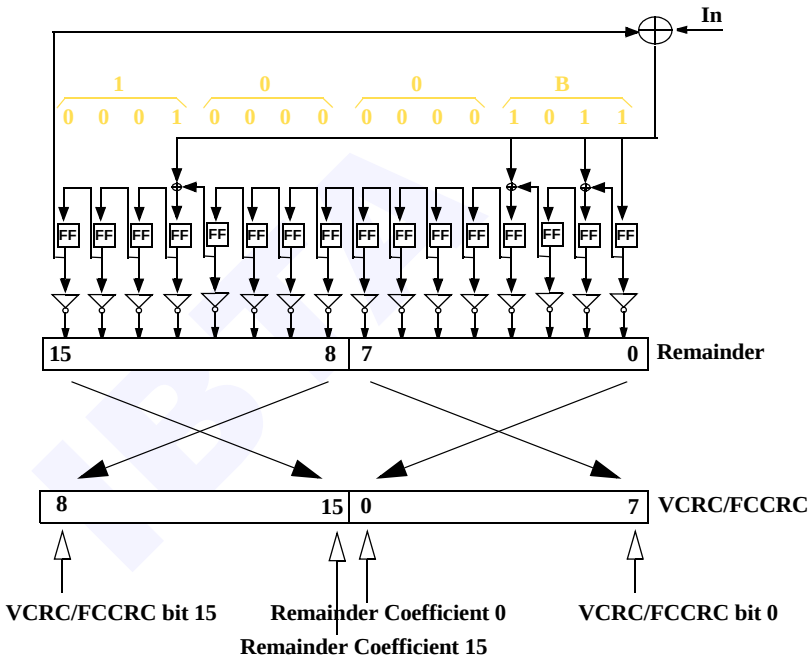


Figure 57 VCRC / FCCRC Generator

7.8.4.3 SAMPLE PACKETS

7.8.4.3.1 LOCAL PACKET EXAMPLE

Figure 58 shows the structure of the local packet used for the example. The packet is a ***RDMA Write Only*** carrying a payload of 14 bytes.

Figure 58 Local Packet Example

The header values for the sample packet are shown in Table 20, Table 21 and Table 22 respectively. The data payload is shown in Table 23.

Table 20 LRH

Field	Value
VL	0x7
LVer	0x0
SL	0x1
LNH	0x2
DLID	0x375C
PktLen	0xE
SLID	0x17D2

Table 21 BTH

Field	Value
Opcode	0x0A
SE	0x0
M	0x0
Pad	0x2
TVer	0x0
PKey	0x2487
Dest QP	0x87B1B3
AckReq	0x0
PSN	0x0DEC2A

Table 22 RETH

Field	Value
VA	0x01710A1C015D4002
RKey	0x38f27A05
DMA Length	0x0000000E

Table 23 Payload

Byte	Value
0	0xBB
1	0x88
2	0x4D
3	0x85
4	0xFD
5	0x5C
6	0xFB
7	0xA4
8	0x72
9	0x8B
10	0xC0
11	0x69
12	0x0E
13	0xD4

The combined byte stream for the Local Packet (before ICRC and VCRC) is shown in Table 24

Table 24 Local Packet Byte Stream (before ICRC and VCRC)

Byte	Value	Byte	Value	Byte	Value	Byte	Value
0	0x70	15	0xB3	30	0x7A	45	0x8B
1	0x12	16	0x00	31	0x05	46	0xC0
2	0x37	17	0x0D	32	0x00	47	0x69
3	0x5C	18	0xEC	33	0x00	48	0x0E
4	0x00	19	0x2A	34	0x00	49	0xD4
5	0x0E	20	0x01	35	0x0E	50	0x00
6	0x17	21	0x71	36	0xBB	51	0x00
7	0xD2	22	0x0A	37	0x88		
8	0x0A	23	0x1C	38	0x4D		
9	0x20	24	0x01	39	0x85		
10	0x24	25	0x5D	40	0xFD		
11	0x87	26	0x40	41	0x5C		
12	0x00	27	0x02	42	0xFB		
13	0x87	28	0x38	43	0xA4		
14	0xB1	29	0xF2	44	0x72		

Table 25 shows the masked byte stream used for ICRC calculation.

Table 25 Masked Byte Stream for ICRC Calculation

Byte	Value	Byte	Value	Byte	Value	Byte	Value
0	0x**F**0	15	0xB3	30	0x7A	45	0x8B
1	0x12	16	0x00	31	0x05	46	0xC0
2	0x37	17	0x0D	32	0x00	47	0x69
3	0x5C	18	0xEC	33	0x00	48	0x0E
4	0x00	19	0x2A	34	0x00	49	0xD4
5	0x0E	20	0x01	35	0x0E	50	0x00
6	0x17	21	0x71	36	0xBB	51	0x00
7	0xD2	22	0x0A	37	0x88		
8	0x0A	23	0x1C	38	0x4D		
9	0x20	24	0x01	39	0x85		
10	0x24	25	0x5D	40	0xFD		
11	0x87	26	0x40	41	0x5C		
12	0xFF	27	0x02	42	0xFB		
13	0x87	28	0x38	43	0xA4		
14	0xB1	29	0xF2	44	0x72		

Generated ICRC is: 0x9625B75A

Generated VCRC is: 0x45FA

Table 26 shows the complete Local Packet Byte Stream.

Table 26 Local Packet Byte Stream

Byte	Value	Byte	Value	Byte	Value	Byte	Value
0	0x70	15	0xB3	30	0x7A	45	0x8B
1	0x12	16	0x00	31	0x05	46	0xC0
2	0x37	17	0x0D	32	0x00	47	0x69
3	0x5C	18	0xEC	33	0x00	48	0x0E
4	0x00	19	0x2A	34	0x00	49	0xD4
5	0x0E	20	0x01	35	0x0E	50	0x00
6	0x17	21	0x71	36	0xBB	51	0x00
7	0xD2	22	0x0A	37	0x88	52	0x96
8	0x0A	23	0x1C	38	0x4D	53	0x25
9	0x20	24	0x01	39	0x85	54	0xB7
10	0x24	25	0x5D	40	0xFD	55	0x5A
11	0x87	26	0x40	41	0x5C	56	0x45
12	0x00	27	0x02	42	0xFB	57	0xFA
13	0x87	28	0x38	43	0xA4		
14	0xB1	29	0xF2	44	0x72		

7.8.4.3.2 GLOBAL PACKET EXAMPLE

Figure 59 shows the structure of the Global packet used for the example.

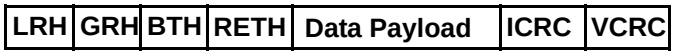


Figure 59 Global Packet Example

The BTH, RETH and data payload for the Global example packet are the same as for the Local packet one. The values for the LRH and GRH fields are shown in Table 27 and Table 28.

.

Table 27 LRH

Field	Value
VL	0x7
LVer	0x0
SL	0x1
LNH	0x3
DLID	0x375C
PktLen	0x18
SLID	0x17D2

Table 28 GRH

Field	Value
IPVer	0x6
TClass	0x00
FlowLabel	0x00000
PayLen	0x0030
NxtHdr	0x00
HopLmt	0x10
SGID	0x00000000000001250000000000000026
DGID	0x00000000000001170000000000000096

The combined byte stream for the Global Packet (before ICRC and VCRC) is shown in Table 29

Table 29 Global Packet Byte Stream (before ICRC and VCRC)

Byte	Value	Byte	Value	Byte	Value	Byte	Value
0	0x70	25	0x00	50	0x24	75	0x0E
1	0x13	26	0x00	51	0x87	76	0xBB
2	0x37	27	0x00	52	0x00	77	0x88
3	0x5C	28	0x00	53	0x87	78	0x4D
4	0x00	29	0x00	54	0xB1	79	0x85
5	0x18	30	0x00	55	0xB3	80	0xFD
6	0x17	31	0x26	56	0x00	81	0x5C
7	0xD2	32	0x00	57	0x0D	82	0xFB
8	0x60	33	0x00	58	0xEC	83	0xA4
9	0x00	34	0x00	59	0x2A	84	0x72
10	0x00	35	0x00	60	0x01	85	0x8B
11	0x00	36	0x00	61	0x71	86	0xC0
12	0x00	37	0x00	62	0x0A	87	0x69
13	0x30	38	0x01	63	0x1C	88	0x0E
14	0x00	39	0x17	64	0x01	89	0xD4
15	0x10	40	0x00	65	0x5D	90	0x00
16	0x00	41	0x00	66	0x40	91	0x00
17	0x00	42	0x00	67	0x02		
18	0x00	43	0x00	68	0x38		
19	0x00	44	0x00	69	0xF2		
20	0x00	45	0x00	70	0x7A		
21	0x00	46	0x00	71	0x05		
22	0x01	47	0x96	72	0x00		
23	0x25	48	0x0A	73	0x00		
24	0x00	49	0x20	74	0x00		

Table 30 shows the masked byte stream used for ICRC calculation.

Table 30 Masked Byte Stream for ICRC Calculation

Byte	Value	Byte	Value	Byte	Value	Byte	Value
0	0x**FF**	25	0x00	50	0x24	75	0x0E
1	0x**FF**	26	0x00	51	0x87	76	0xBB
2	0x**FF**	27	0x00	52	0xFF	77	0x88
3	0x**FF**	28	0x00	53	0x87	78	0x4D
4	0x**FF**	29	0x00	54	0xB1	79	0x85
5	0x**FF**	30	0x00	55	0xB3	80	0xFD
6	0x**FF**	31	0x26	56	0x00	81	0x5C
7	0x**FF**	32	0x00	57	0x0D	82	0xFB
8	0x6**F**	33	0x00	58	0xEC	83	0xA4
9	0x**FF**	34	0x00	59	0x2A	84	0x72
10	0x**FF**	35	0x00	60	0x01	85	0x8B
11	0x**FF**	36	0x00	61	0x71	86	0xC0
12	0x00	37	0x00	62	0x0A	87	0x69
13	0x30	38	0x01	63	0x1C	88	0x0E
14	0x00	39	0x17	64	0x01	89	0xD4
15	0x**FF**	40	0x00	65	0x5D	90	0x00
16	0x00	41	0x00	66	0x40	91	0x00
17	0x00	42	0x00	67	0x02		
18	0x00	43	0x00	68	0x38		
19	0x00	44	0x00	69	0xF2		
20	0x00	45	0x00	70	0x7A		
21	0x00	46	0x00	71	0x05		
22	0x01	47	0x96	72	0x00		
23	0x25	48	0x0A	73	0x00		
24	0x00	49	0x20	74	0x00		

ICRC Result is:0xB67D1ED1

VCRC Result is: 0xB148

Table 31 shows the complete Global Packet Byte Stream.

Table 31 Global Packet Byte Stream

Byte	Value	Byte	Value	Byte	Value	Byte	Value
0	0x70	25	0x00	50	0x24	75	0x0E
1	0x13	26	0x00	51	0x87	76	0xBB
2	0x37	27	0x00	52	0x00	77	0x88
3	0x5C	28	0x00	53	0x87	78	0x4D
4	0x00	29	0x00	54	0xB1	79	0x85
5	0x18	30	0x00	55	0xB3	80	0xFD
6	0x17	31	0x26	56	0x00	81	0x5C
7	0xD2	32	0x00	57	0x0D	82	0xFB
8	0x60	33	0x00	58	0xEC	83	0xA4
9	0x00	34	0x00	59	0x2A	84	0x72
10	0x00	35	0x00	60	0x01	85	0x8B
11	0x00	36	0x00	61	0x71	86	0xC0
12	0x00	37	0x00	62	0x0A	87	0x69
13	0x30	38	0x01	63	0x1C	88	0x0E
14	0x00	39	0x17	64	0x01	89	0xD4
15	0x10	40	0x00	65	0x5D	90	0x00
16	0x00	41	0x00	66	0x40	91	0x00
17	0x00	42	0x00	67	0x02	92	0xB6
18	0x00	43	0x00	68	0x38	93	0x7D
19	0x00	44	0x00	69	0xF2	94	0x1E
20	0x00	45	0x00	70	0x7A	95	0xD1
21	0x00	46	0x00	71	0x05	96	0xB1
22	0x01	47	0x96	72	0x00	97	0x48
23	0x25	48	0x0A	73	0x00		
24	0x00	49	0x20	74	0x00		

7.8.4.3.3 LINK PACKET EXAMPLE

The field values for the Link Packet example are shown in Table 32.

Table 32 Link Packet

Field	Value
Op	0x0
FCTBS	0x10D
VL	0x5

Table 32 Link Packet (Continued)

Field	Value
FCCL	0x21B

Generated FCCRC: 0xF9C9

Table 33 Link Packet Byte Stream

Byte	Value
0	0x01
1	0x0D
2	0x52
3	0x1B
4	0xF9
5	0xC9

7.9 FLOW CONTROL

7.9.1 INTRODUCTION

This section describes the link level flow control mechanism utilized by IBA to prevent the loss of packets due to buffer overflow by the receiver at each end of a link. This mechanism does not describe end to end flow control such as might be utilized to prevent transmission of messages during periods when receive buffers are not posted. See 9.7.7.2 End-to-End (Message Level) Flow Control on page 347 for end to end flow control specification.

Throughout this section, the terms “transmitter” and “receiver” are utilized to describe each end of a given link. The transmitter is the node sourcing data packets. The receiver is the consumer of the data packets. Each end of the link has a transmitter and a receiver.

IBA utilizes an “absolute” credit based flow control scheme. Unlike many traditional flow control schemes which provide incremental updates that are added to the transmitters available buffer pool, IBA receivers provide a “credit limit”. A credit limit is an indication of the total amount of data that the transmitter has been authorized to send since link initialization.

Errors in transmission, in data packets, or in the exchange of flow control information can result in inconsistencies in the flow control state perceived by the transmitter and receiver. The IBA flow control mechanism provides for recovery from this condition. The transmitter periodically sends an indication of the total amount of data that it has sent since link initialization. The receiver uses this data to re-synchronize the state between the receiver and transmitter.

7.9.2 FLOW CONTROL BLOCKS

The term “flow control block”, or simply “block” indicates a quantity of data in a data packet. This quantity is defined to be the size of the data packet in bytes (every byte between the local route header and the variant CRC, inclusive) divided by 64 bytes, and rounded up to the next integral value.

7.9.3 RELATIONSHIP TO VIRTUAL LANES

The flow control algorithm defined in this chapter is applied to each virtual lane independently, except for virtual lane 15 which is not subject to link level flow control.

7.9.4 FLOW CONTROL PACKET

Flow Control Packet - general format

bits / bytes	31-24	23-16	15-8	7-0
0-3	Op	FCTBS	VL	FCCL
4-5	LPCRC			

Figure 60 Flow Control Packet Format

C7-53: Flow control packets shall be sent for each VL except VL15 upon entering the LinkInitialize state. When in the PortStates LinkInitialize, LinkArm or LinkActive, a flow control packet for a given virtual lane shall be transmitted prior to the passing of 65,536 symbol times since the last time a flow control packet for the given virtual lane was transmitted.

C7-54: Flow control packets shall use the format specified in Figure 60 Flow Control Packet Format on page 210.

A symbol time is defined as the time required to transmit an eight bit data quantity onto a physical lane, i.e. for links operating at link speed of 2.5Gb/s, the symbol time is 4nsec independent of the width of the link. Flow control packets may be transmitted as often as necessary to return credits and enable efficient utilization of the link. See Section 7.6.4, “Buffering and Flow Control For Data VLs,” on page 183 for additional information.

7.9.4.1 FLOW CONTROL PACKET FIELDS

7.9.4.1.1 OPERAND (OP) - 4 BITS

The flow control packet is a link packet with one of two Op (operand) values: An operand of 0x0 indicates a normal flow control packet. An operand value of 0x1 indicates a flow control init packet.

C7-55: When in the PortState LinkInitialize, flow control packets shall be sent with the flow control init operand, 0x1.

C7-56: When in the PortStates LinkArm or LinkActive, flow control packets shall be sent with the normal flow control operand, 0x0.

C7-57: All other values of the Op field are reserved for operations that may be defined by IBA in the future. Any packet received with a reserved value shall be discarded.

7.9.4.1.2 FLOW CONTROL TOTAL BLOCKS SENT (FCTBS) - 12 BITS

The FCTBS (Flow Control Total Blocks Sent) field is generated by the transmitter side logic. The calculation for the value of FCTBS is described later.

7.9.4.1.3 FLOW CONTROL CREDIT LIMIT (FCCL) -12 BITS

The FCCL (Flow Control Credit Limit) field is generated by the receiver side logic. The calculation for the value of FCCL is described later.

7.9.4.1.4 VIRTUAL LANE (VL) - 4 BITS

VL (Virtual Lane) is set to the virtual lane to which the FCTBS and FCCL fields apply.

7.9.4.1.5 LINK PACKET CYCLIC REDUNDANCY CHECK (LPCRC) - 16 BITS

LPCRC (Link Packet Cyclic Redundancy Check) field is a 16-bit CRC that covers the first four bytes of the flow control packet. See Section 7.8.3, "Link Packet CRC (LPCRC) - 2 Bytes," on page 198.

7.9.4.2 CALCULATION OF FCTBS

C7-58: Upon transmission of a flow control packet, FCTBS shall be set to the total blocks transmitted in the virtual lane since link initialization.

C7-59: When in the PortState initialize, FCTBS shall be set to zero.

7.9.4.3 CALCULATION OF FCCL

The FCCL calculation is based on a 12-bit Adjusted Blocks Received (ABR) counter maintained for each virtual lane at the receiver.

C7-60: The ABR counter shall be set to zero when in the PortState initialize.

C7-61: Upon receipt of each flow control packet, the ABR shall be set to the value of the FCTBS field.

C7-62: Upon receipt of each data packet, the ABR shall be increased by the blocks received, modulo 4096, except that the ABR shall not be increased for received packets that are discarded due to lack of receive capacity in the receiver.

C7-63: The FCCL field shall be set as specified in Section 7.9.4.3, "Calculation of FCCL," on page 211.

Upon transmission of a flow control packet, FCCL shall be set to one of the following:

- If the current buffer state of the receiver would permit reception of 2048 or more blocks from all combinations of valid IBA packets without discard, then the FCCL shall be set to ABR plus 2048 modulo 4096.
- Otherwise the FCCL shall be set to ABR plus the number of blocks the receiver is capable of receiving from all combinations of valid IBA packets without discard modulo 4096.

The number of blocks the receiver is capable of receiving means the number that the receiver can guarantee to receive without buffer overflow regardless of the sizes of the packets that arrive. If, for example, a receiver is capable of receiving more data when large packets arrive than for small packets, the receiver must use the smaller capacity to calculating FCCL.

This specification does not preclude the reconfiguration of receive buffers while the link is active. Such reconfiguration may result in changes of the FCCL value, including the possibility of reduction of available credit. Also, link errors may cause discrepancies between ABR at the receiver and FCTBS at the transmitter. When this has happened, the next flow control update to the receiver will correct the value of ABR and may result in changes of FCCL which reduce or increase credit. When FCCL is updated, the credit calculation for outgoing data packets should use the new value. Packets that are currently being transmitted or queued may be sent based on the previous FCCL value.

7.9.4.4 TRANSMISSION OF PACKETS

If a data packet is available for transmission:

- Let CR represent the total blocks sent since link initialization plus the number of blocks in the data packet to be transmitted, all modulo 4096.
- Let CL represent the last FCCL received in a flow control packet.

If (CL-CR) modulo 4096 ≤ 2048, then the data packet may be transmitted. If the condition is not true, then the data packet may not be transmitted until the condition becomes true. Flow control packet transmission is not subject to this restriction nor are any packets on virtual lane 15.

C7-64: A non-VL15 data packet may only be sent when there is sufficient credit as determined by the calculation in Section 7.9.4.4, "Transmission of Packets," on page 212.

C7-65: VL15 packets shall not be subject to flow control.

7.10 IBA AND RAW PACKET MULTICAST

7.10.1 OVERVIEW

Multicast is a one-to-many communication paradigm designed to improve the efficiency of communication between a set of end nodes. Figure 61 illustrates an example unreliable multicast IBA operation:

- A packet with PSN = 1129 is received on an IBA routing element (switch or router) port.
- Switches extract the multicast DLID from the LRH to determine if it corresponds to a multicast group. An implementation may maintain this data as part of its internal route table, e.g. a bit-mask which corresponds to the output ports this packet should be forwarded.
- Routers extract the GID from the GRH for IBA multicast or, for raw packet support, examine the IPv6 header or Ethertype within the RWH to determine if the packet corresponds to a multicast group. It uses this information to forward the packet to the next hop(s) to the destination(s).
- Switches or routers replicate the packet (implementation dependent) and forward the packet onto the output port(s).

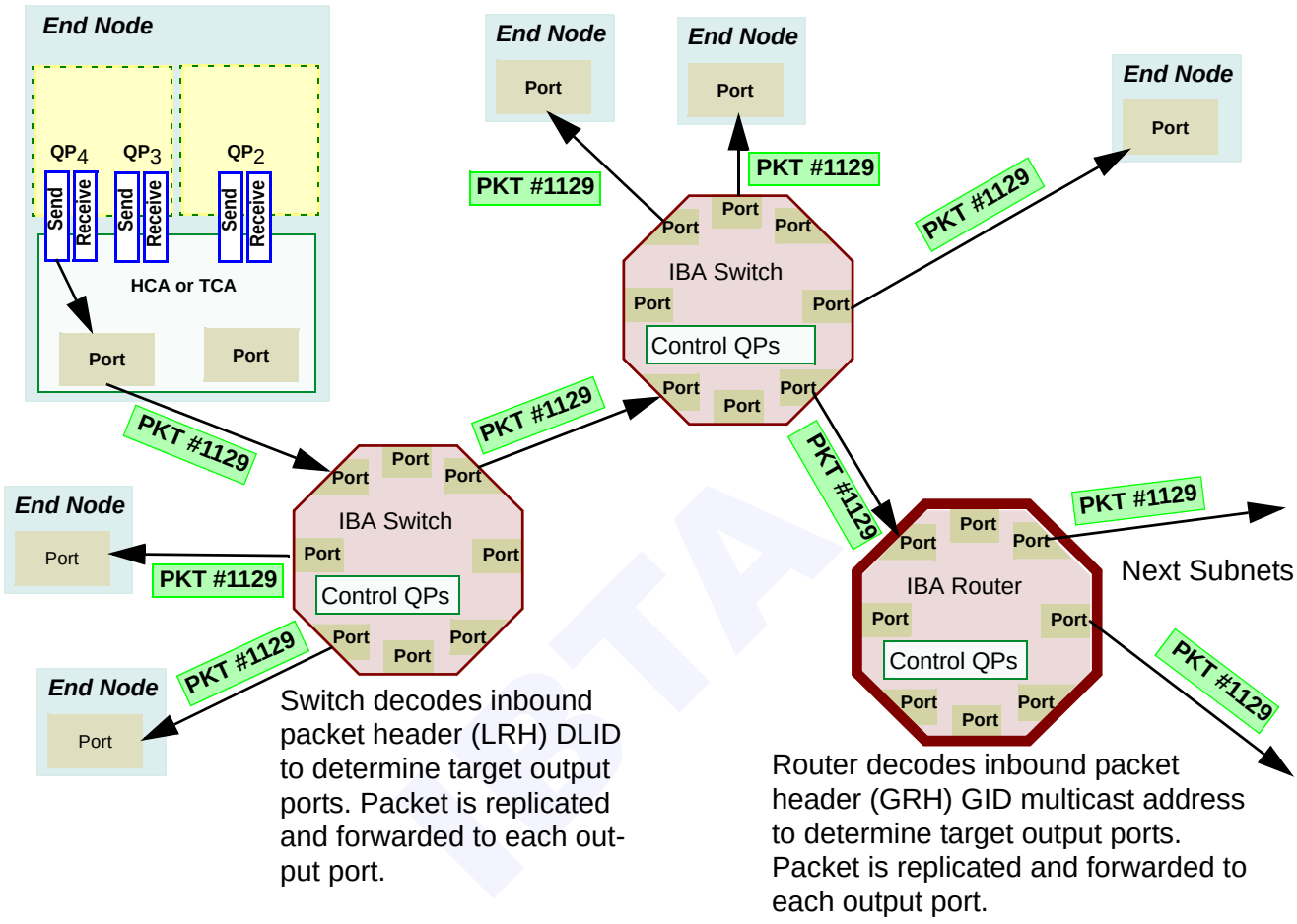


Figure 61 Example IBA Unreliable Multicast Operation

7.10.2 IBA UNRELIABLE MULTICAST OPERATIONAL RULES

o7-13: This compliance statement is obsolete and has been replaced by o7-13.1.1:.

o7-13.1.1: IIBA unreliable multicast is an optional capability. When implemented, it shall function based on the operational rules in Section 7.10.2, "IBA Unreliable Multicast Operational Rules," on page 214.

1) Multicast capability discovery, route table modification, status, and control shall be administered by an IBA management entity. Refer to 15.2.5.8 MulticastForwardingTableRecord on page 893 and 14.2.5.12 MulticastForwardingTable on page 838".

2) Within the network, packets are replicated within IBA switches and routers and forwarded to the corresponding output ports.

3) Packets are not reliable with respect to acknowledgment generation nor delivery guarantees.

4) Switches and routers may vary in their ability to support multicast packets and thus may have implementation-specific scheduling, resource management, and congestion policies which are outside the scope of IBA.

5) Application multicast packets may be transmitted on VLs as assigned via the SL to VL mapping table by the subnet manager. The use of VL 15 for multicast is prohibited.

6) Application multicast packet headers may contain any SL as provided or derived from values provided by the subnet manager.

7) Applications targeting a multicast group use a multicast group GID - each endport participating in a multicast group shall be assigned the corresponding multicast group GID.

8) Each CA, switch or router that supports multicast may participate in zero, one, or many multicast groups.

9) Multicast groups may span multiple subnets - a multicast capable router is required to forward packets to the next hop to the destination.

10) Multicast packets may be generated by an endport.

Multicast group membership is opaque to the participating end nodes, i.e. except as noted in Section 15.2.5.17.5 Querying a Multicast Group on page 915, it is impossible to know which end nodes are participating within a multicast group and whether all participating end nodes within a multicast group reside within a local or remote subnet. Therefore, all IBA multicast packets shall contain a GRH with the destination multicast GID defined per the IBA addressing rules.

11) The SGID within the GRH shall be set to the source port which initially injected the packet into the network.

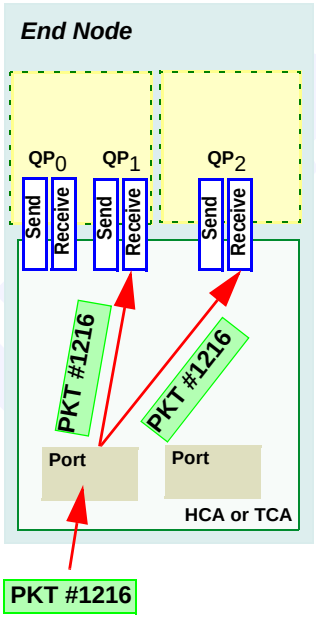
12) Messages shall be limited to single-packet messages. The maximum message size is set during the multicast group's creation. The group creator sets the Path MTU (PMTU) for the multicast group. A CA / router will query the SM for the PMTU during multicast group join operation.

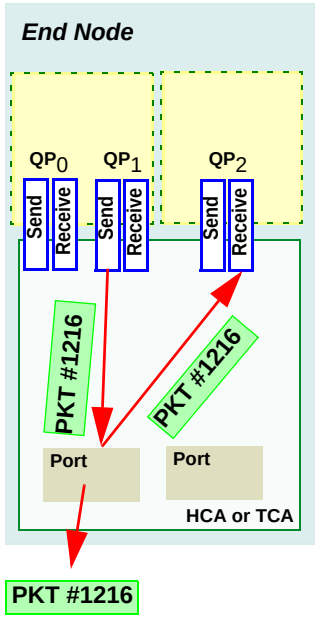
    - If an end node attempts to join a multicast group and is unable to accept the current PMTU, the join operation must fail.

13) For each multicast group an endport is participating in, the port shall associate at least one locally managed QP.

- If a source port is also a destination port within the destination multicast group, the source shall internally replicate the packet within the channel interface to the associated local QPs, including the source QP if it is a receiver for this multicast group.
- If the destination end node contains multiple locally managed QPs participating in a multicast group, the destination end node is responsible for internally replicating the packet within the channel interface and delivering a copy to each QP.

Destination end node delivers one internally replicated copy of the packet to each locally managed QP participating in the indicated multicast group.





If the source end node contains QPs which are targets of send operations, the end node shall internally replicate the packet and deliver it to each participating QP. Replication occurs within the channel interface and may be performed either in hardware or software.

Figure 62 Packet Delivery within an end node

14) Unreliable multicast shall use the unreliable datagram transport service. Refer to the unreliable datagram transport services section for operational rules, constraints, verification, and error handling.

15) A source end node shall set the destination QP within the packet header to 0xFFFFFF.

7.10.3 RAW PACKET MULTICAST

Raw packets may be multicast using the same basic principles as unreliable multicast IBA packets.

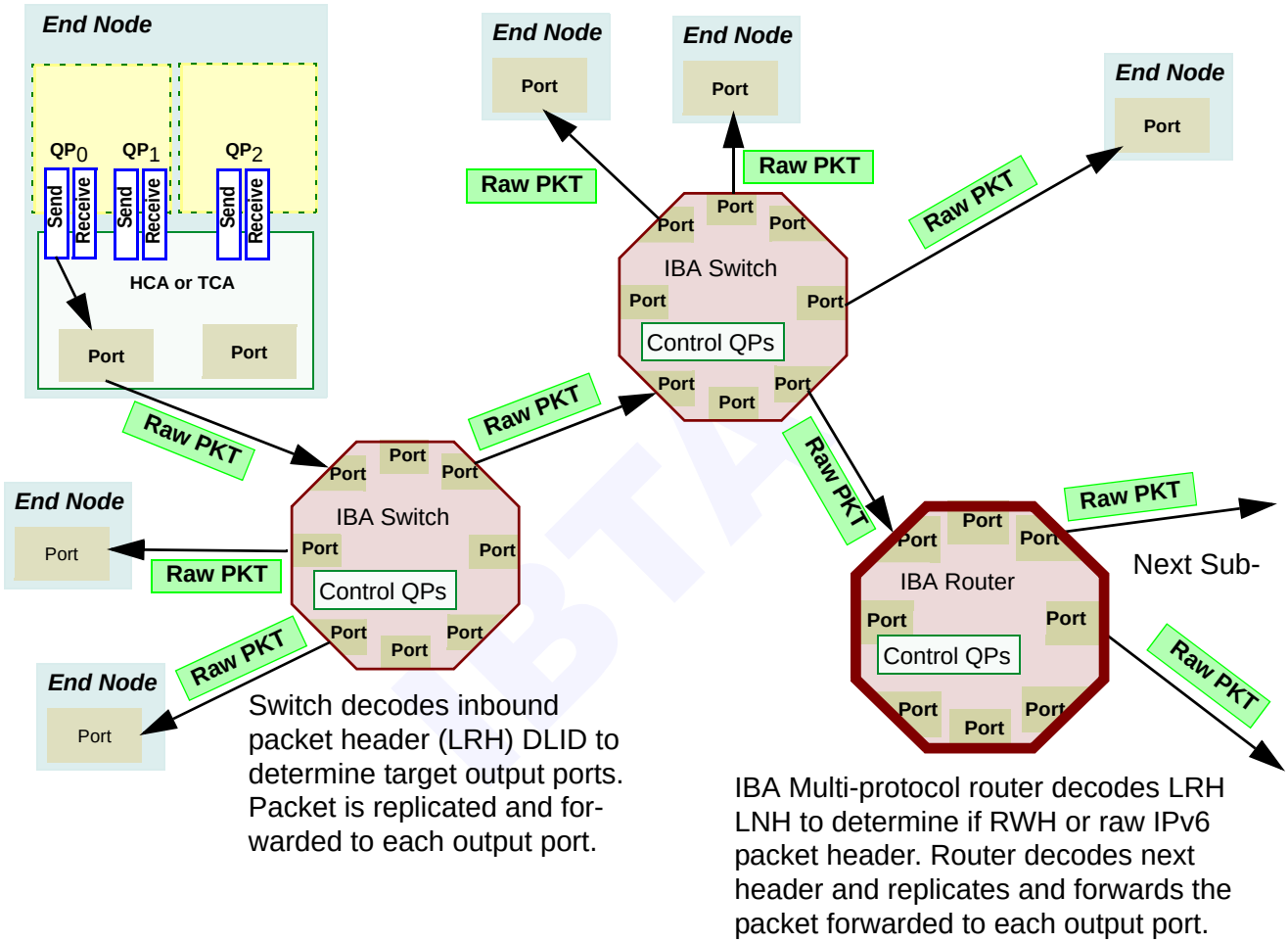


Figure 63 Example Raw Packet Multicast Operation

7.10.3.1 RAW MULTICAST OPERATIONAL RULES

o7-14: Raw packet unreliable multicast is an optional capability. When implemented, it shall function based on the operational rules in Section 7.10.3, "Raw Packet Multicast," on page 217.

1) Raw packet multicast is optional functionality defined within IBA.
2) Raw multicast capability discovery, route table modification, status, and control shall be administered by an IBA management entity.

3) Within the network, packets are replicated within IBA switches and forwarded to the corresponding output ports.

   - Switches extract the multicast DLID from the LRH to determine the corresponding output ports.

4) Routing elements may vary in their ability to support multicast packets and thus may have implementation-specific scheduling, resource management, and congestion / drop policies which are outside the scope of this architecture.

5) Raw multicast packets may be transmitted on any VL except VL 15.

6) Raw multicast packets may be transmitted using any valid SL.

7) IPv6 applications target a multicast group using an IPv6 multicast address. All other protocols use protocol specific addressing and resolution.

8) Each endnode which supports multicast may participate in zero, one, or many multicast groups.

9) Raw multicast groups may span multiple subnets - a multicast capable router is required to forward packets to the next hop to the destination.

10) Raw multicast packets may be generated by an endnode.

11) Messages shall be limited to single-packet messages. The maximum message size is a function of the PMTU between the source and destination end nodes. Raw protocol management will interact with IBA management entity to determine the maximum PMTU allowed. Raw multicast operations are not required to fail if the PMTU is too small - error recovery is the responsibility of the raw multicast group management protocol.

12) Raw packet support requires a minimum of one locally managed QP. An implementation may provide additional QPs based on implementation-specific policies. As such, implementations are responsible for local raw packet replication and delivery.

    - If a source port is also a destination port within the destination multicast group, the source shall internally replicate the packet within the channel interface to the associated local application targets.

    - If the destination end node contains multiple participating application targets within a raw multicast group, the destination end node is responsible for internally replicating the packet within the channel interface and delivering a copy to each target.

13) Raw packet multicast shall use the IBA raw packet header formats and semantics.

7.10.4 GROUP MANAGEMENT

IBA Release 1.1 does not fully define the multicast group management protocol used to implement join and leave operations. However, the management section does contain the management interface and associated MADs to implement a multicast group protocol. Multicast group management is covered in 15.2.5.17 MCMemberRecord on page 908.

7.11 SUBNET MULTIPATHING

7.11.1 MULTIPATHING REQUIREMENTS ON END NODE

Each CA and router port is initialized with a LID plus an LMC (LID Mask Control) by the subnet manager. The value of LMC indicates the number of low order bits of the LID to mask when checking a received DLID against the port's DLID. LMC may take values from 0 to 7. Therefore, a port may be identified by 1 to 128 unicast LIDs.

C7-66: When a link layer of a CA or router port checks that a unicast DLID in a received packet is a valid DLID for that port, it shall mask the number of low order bits indicated by the LMC before comparing the DLID to its assigned LID.

The subnet manager may program alternate paths through the subnet for these various LIDs. The selection of which LID to use in the SLID and DLID of transmitted packets is covered in the Transport chapter.

7.12 ERROR DETECTION AND HANDLING

7.12.1 ERROR DETECTION

The following classes of errors are detectable by the link layer:

- Single packet receive errors
  - Local physical errors - errors indicative of bit errors on the attached physical link. Failures of ICRC, LPCRC and VCRC checks in the packet check state machines and entry to the bad packet state of the packet receiver state machine belong to this class.
  - Remote physical errors - errors indicative of bit errors on a link other than the attached physical link. Entry to the marked bad packet state of the packet receiver state machine belongs to this class.
  - Malformed packet errors - errors indicative of packets transmitted with inconsistent content. The packet was possibly bad at the source. It is also possible that the error was inserted by a switch. Programming errors of switch or port configuration by the SM

may also create errors in this category. LVer error, Length error, op_code error, VL error, and GRH_VL15 error belong to this class. These are all errors from the packet check state machines.

- Switch routing errors - errors indicative of an error in switch routing. DLID errors are in this class.
- Buffer overrun - error indicative of an error in the state of the flow control machine in the link layer at the other end of the physical link. One cause of such an error can be an earlier packet with a physical error if buffers are not immediately reclaimed from bad packets.

C7-67: An error in a received packet shall be classified as specified in Section 7.12.1, “Error Detection,” on page 219.

C7-68: When error counters for the single packet receive errors are implemented and one or more errors are detected in a received packet, then the counter associated with the error with the highest precedence as defined by Section 7.3, “Packet Receiver States,” on page 172, Section 7.4, “Data Packet Check,” on page 175, and Section 7.5, “Link Packet Check,” on page 178 shall increment and none of the other single packet error counters shall increment.

- Receiver errors
  - Local link integrity - excessively frequent local physical errors. This error is caused by a marginal link. A more severe physical problem will be detected at the physical layer based on high frequency of code violations. Detection of local link integrity errors is based on a count of local physical errors. The count starts at zero and shall be incremented for each packet received with a local physical error. If the current count is above zero, the counter shall be decremented once for each packet received without a local physical error. When it exceeds local_phy_errors threshold, the local link integrity error shall be detected.
  - Excessive buffer overruns - buffer overrun errors persisting over multiple flow control update times. This error shall be detected when the number of consecutive flow control update periods with at least one overrun error in each period exceeds the OverrunErrors threshold given in the PortInfo attribute. A flow control update period should correspond to a time interval of 65536 symbol times.

C7-69: This compliance statement is obsolete and has been replaced by C7-69.1.1:.

C7-69.1.1: Each port shall implement detection of local link integrity and excessive buffer overrun errors as specified in Section 7.12.1, “Error Detection,” on page 219

- Transmitter errors
  - Flow control update - errors indicative of a failure of the flow control machine at the other end of the link. For each VL active in the current port configuration, except VL 15 there shall be a watchdog timer monitoring the arrival of flow control updates. If the timer expires without receiving an update, a flow control update error has occurred. The period of the watchdog timer shall be 400,000 +3%/-51% symbol times. This timer shall only run when PortState = Arm or Active. When PortState = ActiveD, this timer shall be reset. When PortState = Initialize or when a flow control packet is received, the timer shall be reset.

C7-70: Each port shall implement detection of flow control update errors as specified in Section 7.12.1, “Error Detection,” on page 219.

7.12.2 ERROR RECOVERY PROCEDURES

The response to any single packet receive error is to discard the packet. No further recovery is necessary at the link layer. For some errors, the data packet check state machine (Section 7.4, “Data Packet Check,” on page 175) allows a switch to forward a packet with an error marking it as bad by appending a bad VCRC value and the EBP delimiter as an alternative to dropping the packet.

Local link integrity, excessive buffer overrun, and flow control update errors all indicate errors that may be fixed by retraining or may be due to a hard fault.

C7-71: Upon detecting local link integrity, excessive buffer overrun or flow control update errors, the link shall initiate retraining by asserting L_init_train (refer to 6.3.1.2 L_Init_Train - Link Initiate Retraining on page 164).

7.12.3 ERROR NOTIFICATION

Single packet receive error classes increment error counters as specified in management (Refer to 16.1.4 Optional Attributes on page 950). Note that at most one link layer error is detected per packet so each packet increments one and only one of these counters.

Local link integrity, excessive buffer overrun, and flow control update are counted and may produce a trap as specified in management.

CHAPTER 8: NETWORK LAYER

8.1 OVERVIEW

This chapter describes the network layer within IBA. Within the IBA layered architecture, this layer is responsible for routing packets between IBA subnets. This includes unicast and multicast operations. This chapter specifies routing between IBA subnets - it does not specify multi-protocol routing, i.e. routing IBA over non-IBA fabric types, nor does it specify how raw packets are routed between IBA subnets.

This chapter, with the exception of section *8.4 Global Route Header Usage on page 226*, is informational in nature. As such, it does not specify IBA requirements; refer to Chapter 19: Routers on page 1059 for requirements of IBA routers. Packet forwarding within an IBA subnet is done at the link layer by IBA switches; refer to Chapter 18: Switches on page 1040 for requirements of IBA switches.

8.2 PACKET ROUTING

8.2.1 OVERVIEW

IBA supports a two-layer topological division. The lower layer is referred to as an IBA subnet. Packets are forwarded throughout the subnet utilizing IBA switches (the process of forwarding a packet from one link to another within a subnet is referred to as switching). The path that a packet takes through this layer is uniquely defined by its point of injection into the fabric, identified in the packet by the SLID field in the LRH, and the DLID and SL fields in its LRH.

At the higher layer, subnets are interconnected using routers (the process of forwarding a packet from one subnet to another is referred to as routing). Routing may be accomplished utilizing routers conforming to the IBA specification, and may also be accomplished using routers conforming to other specifications (e.g. utilizing the Internet Protocol (IP) suite of specifications). The series of subnets through which a packet passes is not defined by IBA; however, several fields are provided in the Global Route Header to enable routers to make this decision. These fields include SGID, DGID, TClass and FlowLabel. Additionally, a router might use fields from other headers, e.g. the SL field in the LRH to determine a mapping to TClass. Regardless of the mechanism used to in forwarding decisions, IBA requires that the path be symmetric with respect to SGID and DGID. This means that if a valid path exists from an SGID to a DGID, then IBA requires that a valid path also exist swapping the values of DGID and SGID.

The requirements of IBA routers are specified in Chapter 19: Routers on page 1059. Interconnection of IBA subnets utilizing IBA routers is intended to preserve IBA intra-subnet behavior across subnets.

Use of other routing technologies is beyond the scope of IBA; however, the architecture is intentionally crafted to enable this capability, especially utilizing IP version 6 as specified by IETF RFC 2460 and other associated IETF RFCs.

A global IBA fabric consists of one IBA subnet or multiple IBA subnets interconnected via routers. As described above, this global fabric may also include non-IBA interconnects between IBA subnets, as well as gateways to non-IBA fabrics.

8.2.2 GLOBAL FABRIC CHARACTERISTICS

This section describes the characteristics of a global fabric interconnected exclusively with IBA routers. While beyond the scope of IBA, global fabrics interconnected with non-IBA technology may also exhibit some or all of these characteristics.

8.2.2.1 INHERITANCE OF SUBNET REQUIREMENTS

All the packet delivery characteristics of a subnet are inherited by the global fabric, except for virtual lane 15 subnet management packets (since subnet management occurs at the subnet level, these packets do not transit routers).

8.2.2.2 PACKET ERRORS AND ERROR DETECTION

IBA specifies an invariant CRC that is appended to all IBA packets except raw packets (refer to section 7.8.1 Invariant CRC (ICRC) - 4 Bytes on page 195). This CRC covers all of the IBA packet fields that do not require modification to effect IBA routing. End-to-end data integrity assurance is provided by retaining this CRC unmodified as the packet transits the global fabric.

8.2.2.3 SERVICE LEVELS

Service levels and virtual lanes are supported throughout the global fabric. This is accomplished by mapping service level to traffic class in the GRH, and vice versa. The mapping function itself, as is the interpretation of service level, is beyond the scope of IBA.

8.2.3 SUPPORT FOR MULTIPLE GLOBAL PATHS

The information required to route a packet within a subnet and between subnets is contained in the packet's local route header and global route headers, respectively. Unlike many network protocols, IBA does not require a packet to contain a global route header unless the packet is either destined for a device that is not on the same subnet or the packet is a mul-

ticast packet. However, any packet except subnet management packets may contain a global route header (subnet management packets are defined in 14.2.1 Datagram Formats and Use on page 795.)

The identification and utilization of multiple paths between two channel adapters on different subnets is hierarchical and involves similar but independent mechanisms within subnets and across subnets.

Within subnets, multiple paths between two channel adapters are identified by multiple LIDs. That is, a port may effectively be assigned multiple LIDs using a LID/LMC combination Chapter 4: Addressing on page 141. The source channel adapter indicates a path via its selection of one of the LIDs assigned to the destination port.

Likewise, channel adapters have the option to support the assignment of multiple GIDs. In the case of global routing across subnets, the LID indicates which of the valid paths is to be used within the subnet (i.e. switch forwarding) and the GID indicates which of the valid paths is to be used between subnets (i.e. router forwarding).

As a packet transits a subnet, its SLID and DLID fields remain unchanged. As a packet transits between subnets (i.e. through a router), the router updates the SLID to that of its own LID and the DLID to the LID of the next router or final destination, as appropriate.

Note that for global routing, this provides two degrees of freedom for a source channel adapter to select a path through the fabric. Selection of the LID determines the route through the subnet to the first router. Selection of the GID determines the route taken after reaching the first router. Each router along the path may choose the path through a subnet to the next router (or final destination) via its selection of the LID for the next router (or final destination). Furthermore, since the DLID may contain LMC bits of multipath data, the router may use the DLID as part of its route determination algorithm.

The decision process that routers use for forwarding packets is not specified by IBA; however, routers may rely on various combinations of Destination GID, Source GID, SL, TClass, and FlowLabel fields, among other factors, to determine the forwarding path and flows that must exhibit in-order delivery. Channel Adapters and/or ingress routers may label flows of packets that are expected to be delivered in order with the same FlowLabel in the global route header. While IBA routers utilize LIDs and GIDs to determine paths, the FlowLabel may be used by non-IBA routers to determine paths.

8.2.4 GLOBAL MULTICAST

IBA supports an unreliable multicast mechanism. A detailed description of this mechanism may be found in section 7.10 IBA and Raw Packet Multicast on page 213. Implementation of this mechanism is optional in IBA devices (including switches and routers). Multicast packets within a given multicast group, i.e. multicast packets that share a common multicast GID, may be sourced by a single device or by multiple devices. Since routers are not fully specified by IBA, routers may vary in their ability to support multicast packets and may have implementation specific.

8.3 GLOBAL ROUTE HEADER

Figure 64 on page 225 illustrates the format of the Global Route Header that is used for inter-subnet routing.

bits bytes	31-24	23-16	15-8	7-0
0-3	IPVer	TClass	FlowLabel	
4-7	PayLen		NxtHdr	HopLmt
8-11	SGID[127-96]			
12-15	SGID[95-64]			
16-19	SGID[63-32]			
20-23	SGID[31-0]			
24-27	DGID[127-96]			
28-31	DGID[95-64]			
32-35	DGID[63-32]			
36-39	DGID[31-0]			

Figure 64 Global Route Header (GRH)

Global route headers are not required in all packets (see section 8.4.1 Global Route Header Generation on page 226 for details). The presence of a Global Route Header is indicated in the Local Route Header as specified in 7.7.5 Link Next Header (LNH) - 2 bits on page 194. The following subparagraphs describe the fields in the GRH:

8.3.1 IP VERSION (IPVER) - 4 BITS

Indicates the version of the GRH; always set to 6.

8.3.2 TRAFFIC CLASS (TCLASS) - 8 BITS

This field is used to communicate service level end-to-end, i.e. across subnets. The mapping of specific traffic class to specific TClass values is not specified by IBA and may vary by implementation.

8.3.3 FLOW LABEL (FLOWLABEL) - 20 BITS

This field may be used to identify a sequence of packets that must be delivered in order.

8.3.4 PAYLOAD LENGTH (PAYLEN) - 16 BITS

For an IBA packet this field specifies the number of bytes starting from the first byte after the GRH up to and including the last byte of the ICRC. For a raw IPv6 datagram this field specifies the number of bytes starting from the first byte after the GRH up to but not including either the VCRC or any padding to achieve a multiple of a 4 byte packet length.

8.3.5 NEXT HEADER (NXTHDR) - 8 BITS

This field indicates what header, if any, follows the global route header.

8.3.6 HOP LIMIT (HOPLMT) - 8 BITS

This field indicates the number of hops (i.e. the number of routers transited) that the packet is permitted to take prior to being discarded. This ensures that a packet will not loop indefinitely between routers should a routing loop occur. Setting this value to 0 or 1 will ensure that the packet will not be forwarded beyond the local subnet.

8.3.7 SOURCE GLOBAL IDENTIFIER (SGID) - 128 BITS

This field identifies the port that injected the packet into the global fabric. Additional information on the format and use of GID's may be found in Chapter 4: Addressing on page 141.

8.3.8 DESTINATION GLOBAL IDENTIFIER (DGID) - 128 BITS

This field identifies the final destination port of the packet, or to the multicast group that represents the set of ports to which the packet is to be delivered. Additional information on the format and use of GID's may be found in Chapter 4: Addressing on page 141.

8.4 GLOBAL ROUTE HEADER USAGE

The following subsections describe the usage of the global route header:

8.4.1 GLOBAL ROUTE HEADER GENERATION

C8-1: A channel adapter initiating a packet shall include a global route header if any of the following conditions apply:

- The packet is a multicast packet.
- The final destination of the packet is a port of a device that is not on the same subnet as the port that initially injects the packet into the fabric and both the injecting and receiving ports are connected to IBA subnets.

o8-1: A channel adapter, switch, or router initiating a packet may include a global route header in any packet except for SMPs.

The use of a global route header should be negotiated during connection establishment time. For unreliable datagram services (not including multicast which requires a global router header for all multicast packets), the process to determine whether to use a global route header or not is outside the scope of the specification.

If a global route header is included, the fields are loaded by the initiating channel adapter, switch, or router as follows:

C8-2: IPVer: If a global route header is included in a packet, this field shall be set to 6.

C8-3: TClass: If a global route header is included in a packet, this field shall either be set to zero or to an appropriate TClass value by the injecting channel adapter. Each router maps TClass to a SL appropriate for the subnet on which it will inject the packet. This mapping function is not specified by IBA.

FlowLabel: The use of this field is not required by IBA.

C8-4: If a global route header is included in a packet, and FlowLabel is not used, it shall be set to zero.

C8-5: If a global route header is included in a packet and FlowLabel is used, all packets that must be delivered in order with respect to each other shall be identified by a constant, non-zero value inserted in the FlowLabel field.

This implies that if a given QP uses a non-zero flow label, it must use the same flow label on all packets emitted from that QP that are destined for a given remote QP. Different QPs transmitting to a given destination may use the same or different flow labels. Flow labels may be shared among QPs.

C8-6: PayLen: If a global route header is included in an IBA packet, this field shall be loaded with the length of the packet, in bytes, starting from the first byte after the global route header up to and including the last byte of the ICRC.

NxtHdr: The use of this field varies depending on whether the packet is a raw or non-raw packet.

C8-7: For non-raw IBA packets that include a GRH, the NxtHdr field shall contain *0x1B*.

C8-8: For raw packets that include a IPv6 header, the contents of NxtHdr shall be set to the identifier for the next header as defined in IETF RFC 1700 et. seq.

C8-9: HopLmt: If a global route header is included in a packet, this field shall be set to the number of hops (i.e. the number of routers that may be transited) that the packet is permitted to take prior to being discarded.

C8-10: SGID: If a global route header is included in a packet, this field shall be set to one of the GID's assigned to the port that will inject the packet into the fabric.

C8-11: DGID: If a global route header is included in a packet, this field shall be set to one of the GID's assigned to the port that is the final destination of this packet, or to the multicast GID that represents the set of ports to which the packet is to be delivered.

8.4.2 GLOBAL ROUTE HEADER MODIFICATION

This section describes the modifications that may and must be made to the global route header by IBA routers when forwarding packets between subnets. Note that modification of these fields implies updating the packet's variant CRC defined in 7.8.2 Variant CRC (VCRC) - 2 Bytes on page 197. These changes do not affect the packet's invariant CRC defined in 7.8.1 Invariant CRC (ICRC) - 4 Bytes on page 195.

C8-12: IPVer: This field shall not be changed by IBA routers.

TClass: This field is used to communicate service level end-to-end, i.e. across subnets. Routers utilize this field to determine an appropriate SL for forwarding on the next subnet. This mapping function is not specified by IBA.

C8-13: The TClass field, if non-zero, shall not be modified by IBA routers.

The use of TClass by routers when it contains zero is not defined by IBA.

FlowLabel: This field may be used to identify a sequence of packets that must be delivered in order. The use of this field is not required by IBA. If not used, it is left unchanged. If used, all packets that must be delivered in order with respect to each other shall be identified by a constant, non-zero value inserted in this field in each packet.

o8-2: The router may change the value of FlowLabel; however, it must use the same flow label for all packets that must be delivered in order, which includes all traffic between any given two QPs.

C8-14: PayLen: IBA routers shall not modify the content of PayLen.

C8-15: NxtHdr: IBA routers shall not modify the content of NxtHdr.

C8-16: HopLmt: IBA routers shall discard packets that contain a value of one or zero in the HopLmt field. Otherwise, IBA routers shall decrement the HopLmt field by one.

C8-17: SGID: IBA routers shall not modify the content of SGID

C8-18: DGID: IBA routers shall not modify the content of DGID.

8.4.3 GLOBAL ROUTE HEADER VERIFICATION

Any required checks on DGID are performaed at the transport layer, see Section 9.6.1.2 GRH Checks

C8-19: This compliance statement is obsolete and has been removed.

CHAPTER 9: TRANSPORT LAYER

9.1 OVERVIEW

Each IBA packet contains a transport header. The transport header contains the information required by the endnode to complete the specified operation, e.g. delivery of data payload to the appropriate entity within the endnode such as a thread or IO controller. This chapter defines the transport services used by IBA.

The client of an IBA channel adapter communicates with the transport layer by manipulating a “queue pair” (QP) made up of a Send work queue and a Receive work queue. For a host platform, the client of the transport layer is the Verbs software layer. The client posts buffers or commands to these queues and hardware transfers data from or into the buffers. Throughout this chapter, a QP that initiates an operation, i.e. injects a message into the fabric, is referred to as the requester and the QP that receives the message is referred to as the responder.

When a QP is created, it is associated with one of four IBA transport service types or non-IBA protocol encapsulation services. The transport service describes the degree of reliability and to what and how the QP transfers data.

The four IBA transport service types are:

1) Reliable Connection
2) Reliable Datagram
3) Unreliable Datagram
4) Unreliable Connection

The non-IBA protocol encapsulation services are:

1) Raw IPv6 Datagram
2) Raw Ethertype Datagram

Table 314 Channel Adapter Attributes on page 1025 lists which of these services are required for Host Channel Adapters and Target Channel Adapters. Table 34 below compares several key attributes of these five transport service types.

Reliable transport services use a combination of sequence numbers and acknowledgment messages (ACK / NAK) to verify packet delivery order, prevent duplicate packets and out-of-sequence packets from being pro-

cessed, and to detect missing packets. Upon error detection, e.g. a missing packet, the missing packet along with all subsequent packets will be retransmitted by the requestor. IBA does not support selective packet retransmission nor the out-of-order reception of packets.

An IBA operation is defined to include a request message and, for reliable services, its corresponding response. Thus, the request message is generated by a requester, and a response, if one exists, is generated by the responder.

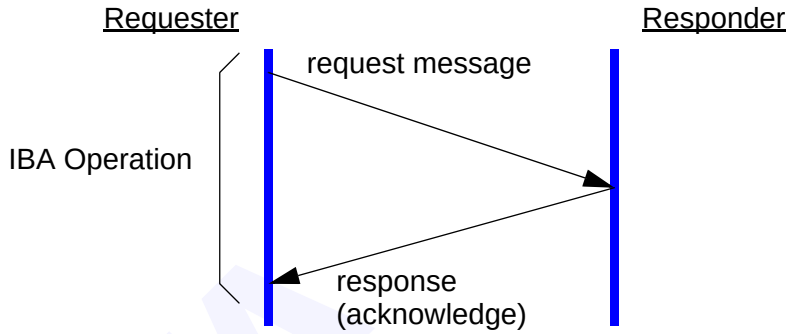


Figure 65 IBA Operation

A request message consists of one or more IBA packets. The packets of a request message are called request packets. A response, except for an RDMA READ Response, consists of exactly one packet. A response is also called an acknowledge. The response packet acknowledges receipt of one or more packets. The response may acknowledge the receipt of packets that comprise anywhere from a portion of a request message to multiple request messages.

Unreliable transport services do not use acknowledgment messages. They do however generate sequence numbers. This allows a responder to detect out-of-sequence or missing packets and to perform local re-

covery processing. The specifics of any recovery processing for unreliable datagrams are outside the scope of the IBA specification.

Table 34 Comparison of IBA Transport Service Types

Attribute	Reliable Connection	Reliable Datagram	Unreliable Datagram	Unreliable Connection	Raw Datagram (both IPv6 & ethertype)
Scalability (M processes on N Processor nodes communicating with all processes on all nodes)	**M^2*N** QPs required on each processor node, per CA	**M** QPs required on each processor node, per CA.	**M** QPs required on each processor node, per CA.	**M^2*N** QPs required on each processor node, per CA.	Minimum **1** QP required on each end node, per CA.

Table 34 Comparison of IBA Transport Service Types (Continued)

Attribute		Reliable Connection	Reliable Datagram	Unreliable Datagram	Unreliable Connection	Raw Datagram (both IPv6 & ethertype)
Reliability	Corrupt data detected	Yes				
	Data delivery guarantee	Data delivered exactly once		No guarantees		
	Data order guaranteed	Yes, per connection	Yes, packets from any one source QP are ordered to multiple destination QPs.	No	Unordered and duplicate packets are detected.	No
	Data loss detected	Yes		No	Yes	No
	Error recovery	**Reliable**. Errors are detected at both the requestor and the responder. The requestor can transparently recover from errors (retransmission, alternate path, etc.) without any involvement of the client application. QP processing is halted only if the destination is inoperable or all fabric paths between the channel adapters have failed.		**Unreliable**. Packets with some types of errors may not be delivered. Neither source nor destination QPs are informed of dropped packets.	**Unreliable**. Packets with errors, including sequence errors, are detected and may be logged by the responder. The requestor is not informed.	**Unreliable**. Packets with errors are not delivered. The requestor and responder are not informed of dropped packets.
RDMA and ATOMIC Operations		Yes	Yes	No	Yes: RDMA WRITEs No: RDMA READs & ATOMICs	No
Bind Memory Window		Yes	Yes	No	Yes	No
IBA Unreliable Multicast Support		No	No	Yes	No	No
Raw Multicast		No	No	No	No	Yes
Remote Invalidation		Yes	No	No	No	No
Shared Receive Queue		Yes	No	Yes	No	No
Message Size		Transport supports a message size of zero to 2^{31} bytes. Implementations may support a smaller maximum message size. Actual maximum message size to be used may be negotiated by upper (application) layers. A message may consist of multiple packets.		**Single PMTU packet** datagrams - 0 to 4096 bytes.	Transport supports a message size of zero to 2^{31} bytes. Implementations may support a smaller maximum message size. Actual maximum message size to be used may be negotiated by upper (application) layers. A message may consist of multiple packets.	**Single PMTU packet** datagrams - 0 to 4096 bytes.

Table 34 Comparison of IBA Transport Service Types (Continued)

Attribute	Reliable Connection	Reliable Datagram	Unreliable Datagram	Unreliable Connection	Raw Datagram (both IPv6 & ethertype)
Connection Oriented?	**Connected**. The client connects the local QP to one and only one remote QP. No other traffic flows over these QPs.	**Connectionless**. Appears connectionless to the client - uses one or more End-to-End contexts per CA to provide reliability service.	**Connectionless**. No prior connection is needed for communication.	**Connected**. The client connects the local QP to one and only one remote QP. No other traffic flows over these QPs.	**Connectionless**. No prior connection is needed for communication.

9.2 BASE TRANSPORT HEADER

Base Transport Header (BTH) contains fields always present for all IBA transport services - it is not present in Raw packets. The presence of BTH is indicated by the Link Next Header (LRH:LNH) field.

C9-1: All IBA transport services shall include a Base Transport Header (e.g. it is not present in Raw packets).

bits / bytes	31-24	23-16	15-8	7-0
0-3	OpCode	SE, M, Pad, TVer	Partition Key	
4-7	Reserved 8 (masked in ICRC)	Destination QP		
8-11	A, Reserved 7	PSN - Packet Sequence Number		

Figure 66 Base Transport Header (BTH)

9.2.1 OPERATION CODE (OPCODE)

The OpCode field defines the interpretation of the remaining header and payload bytes. The OpCode list definition is shown in Table 35 OpCode field on page 235.

C9-2: Table 35 shall be used to define the OpCode parameter in the BTH as well as the headers and payload that follow the BTH.

o9-0.2.1: Any HCA or TCA which implements Remote Invalidate shall use one of the two opcodes defined in Table 35 in the opcode field of the BTH whenever it executes a SEND with Invalidate operation

Table 35 OpCode field

Code[7-5]	Code[4-0]	Description	Packet Contents following the Base Transport header[a]
000 Reliable Connection (RC)	00000	SEND First	PayLd
	00001	SEND Middle	PayLd
	00010	SEND Last	PayLd
	00011	SEND Last with Immediate	ImmDt, PayLd
	00100	SEND Only	PayLd
	00101	SEND Only with Immediate	ImmDt, PayLd
	00110	RDMA WRITE First	RETH, PayLd
	00111	RDMA WRITE Middle	PayLd
	01000	RDMA WRITE Last	PayLd
	01001	RDMA WRITE Last with Immediate	ImmDt, PayLd
	01010	RDMA WRITE Only	RETH, PayLd
	01011	RDMA WRITE Only with Immediate	RETH, ImmDt, PayLd
	01100	RDMA READ Request	RETH
	01101	RDMA READ response First	AETH, PayLd
	01110	RDMA READ response Middle	PayLd
	01111	RDMA READ response Last	AETH, PayLd
	10000	RDMA READ response Only	AETH, PayLd
	10001	Acknowledge	AETH
	10010	ATOMIC Acknowledge	AETH, AtomicAckETH
	10011	CmpSwap	AtomicETH
	10100	FetchAdd	AtomicETH
	10101	Reserved	Undefined
	10110	SEND Last with Invalidate	IETH, PayLd
	10111	SEND Only with Invalidate	IETH, PayLd
	11000-11111	Reserved	undefined

Table 35 OpCode field (Continued)

Code[7-5]	Code[4-0]	Description	Packet Contents following the Base Transport header[a]
001 Unreliable Connection (UC)	00000	SEND First	PayLd
	00001	SEND Middle	PayLd
	00010	SEND Last	PayLd
	00011	SEND Last with Immediate	ImmDt, PayLd
	00100	SEND Only	PayLd
	00101	SEND Only with Immediate	ImmDt, PayLd
	00110	RDMA WRITE First	RETH, PayLd
	00111	RDMA WRITE Middle	PayLd
	01000	RDMA WRITE Last	PayLd
	01001	RDMA WRITE Last with Immediate	ImmDt, PayLd
	01010	RDMA WRITE Only	RETH, PayLd
	01011	RDMA WRITE Only with Immediate	RETH, ImmDt, PayLd
	01100-11111	Reserved	undefined

Table 35 OpCode field (Continued)

Code[7-5]	Code[4-0]	Description	Packet Contents following the Base Transport header[a]
010 Reliable Datagram (RD)	00000	SEND First	RDETH, DETH, PayLd
	00001	SEND Middle	RDETH, DETH, PayLd
	00010	SEND Last	RDETH, DETH, PayLd
	00011	SEND Last with Immediate	RDETH, DETH, ImmDt, PayLd
	00100	SEND Only	RDETH, DETH, PayLd
	00101	SEND Only with Immediate	RDETH, DETH, ImmDt, PayLd
	00110	RDMA WRITE First	RDETH, DETH, RETH, PayLd
	00111	RDMA WRITE Middle	RDETH, DETH, PayLd
	01000	RDMA WRITE Last	RDETH, DETH, PayLd
	01001	RDMA WRITE Last with Immediate	RDETH, DETH, ImmDt, PayLd
	01010	RDMA WRITE Only	RDETH, DETH, RETH, PayLd
	01011	RDMA WRITE Only with Immediate	RDETH, DETH, RETH, ImmDt, PayLd
	01100	RDMA READ Request	RDETH, DETH, RETH
	01101	RDMA READ response First	RDETH, AETH, PayLd
	01110	RDMA READ response Middle	RDETH, PayLd
	01111	RDMA READ response Last	RDETH, AETH, PayLd
	10000	RDMA READ response Only	RDETH, AETH, PayLd
	10001	Acknowledge	RDETH, AETH
	10010	ATOMIC Acknowledge	RDETH, AETH, AtomicAckETH
	10011	CmpSwap	RDETH, DETH, AtomicETH
	10100	FetchAdd	RDETH, DETH, AtomicETH
	10101	RESYNC	RDETH, DETH
	10110-11111	Reserved	undefined
011 Unreliable Datagram (UD)	00000-00011	Reserved	undefined
	00100	SEND only	DETH, PayLd
	00101	SEND only with Immediate	DETH, ImmDt, PayLd
	00110-11111	Reserved	undefined
100 - 101	00000-11111	Reserved	undefined
110 - 111	00000-11111	Manufacturer Specific OpCodes	undefined

a. All OpCodes have the ICRC and VCRC attached.

9.2.2 RESERVED TRANSPORT FUNCTION OPCODES

For future expansion of its transport layer, IBA provides Reserved and Manufacturer Defined BTH OpCodes. Two blocks of undefined OpCodes are specified: one for future revisions of the IBA and one block for manufacturer specific functions. Manufacturer Defined opcodes should not be used between devices until the devices are clearly identified as supporting those opcodes.

9.2.3 SOLICITED EVENT (SE) - 1 BIT

The requester sets this bit to 1 to indicate that the responder shall invoke the CQ event handler. Additional operational guidelines:

- The SE bit should only be set in the last or only packet of a SEND, SEND with Immediate, or RDMA WRITE with Immediate.
- For additional operational guidelines impacting HCAs, see Section 11.4.2.2 Request Completion Notification on page 627.

SE bit is not considered a part of packet header validation, i.e. receipt of a packet with this bit set that does not meet the invocation requirements will not result in a NAK being generated.

C9-3: For an HCA, if an inbound request packet has the Solicited Event bit in the BTH to 1 and the additional SE operational guidelines are valid, it shall invoke the CQ event handler.

o9-1: For a TCA supporting Solicited Events, if an inbound request packet has the Solicited Event bit in the BTH to 1 and the additional SE operational guidelines are valid, it shall invoke the CQ event handler.

C9-4: The responder shall not consider the SE bit in the BTH part of the packet header validation.

In addition to its use in SEND, SEND with Immediate or RDMA Write with Immediate operations, the SE bit can be set with SEND with Invalidate operations. In such a case the SE bit should only be set in the last or only packet of a SEND with Invalidate. In all other respects, the use of the SE bit follows the same rules as defined for the use of the SE bit with a normal SEND operation.

9.2.4 MIGREQ (M) - 1 BIT

Used to communicate migration state. If set to one, indicates the connection or EE context has been migrated; if set to zero, it means there is no change in the current migration state. See Automatic Path Migration within the Chapter 17: Channel Adapters on page 1016.

9.2.5 PAD COUNT (PADCNT) - 2 BITS

Packet payloads are sent as a multiple of 4-byte quantities. Pad count indicates the number of pad bytes - 0 to 3 - that are appended to the packet payload. Pads are used to “stretch” the payload (payloads may be zero or more bytes in length) to be a multiple of 4 bytes.

9.2.6 TRANSPORT HEADER VERSION (TVER) - 4 BITS

Specifies the version of the IBA Transport used for this packet. This version applies to all of the transport fields including the BTH, extended

header and the invariant CRC - this field is set to 0x0. If a receiver does not support the Transport Version specified then the packet is discarded.

C9-5: Requesters and responders using IBA transports shall generate IBA transport packets with BTH:TVer = 0x0.

9.2.7 PARTITION KEY (P_KEY) - 16 BITS

P_Key identifies the partition that the destination QP (RC, UC, UD) or EE Context (RD) is a member.

9.2.8 DESTINATION QP (DESTQP) - 24 BITS

This field specifies the destination queue pair (QP) identifier.

9.2.9 RESERVE 8 (RESV8) - 8 BITS

Reserved (variant) - 8 bits. Transmitted as 0, ignored on receive. This field is not included in the invariant CRC.

C9-6: When generating a packet, the sender shall set the Resv8 field to zero. The receiver shall ignore this field.

9.2.10 ACKREQ (A) - 1 BIT

Requests responder to schedule an acknowledgment on the associated QP.

9.2.11 RESERVE 7 (RESV7) - 7 BITS

Transmitted as 0, ignored on receive. This field is included in the invariant CRC.

C9-7: When generating a packet, the sender shall set the Resv7 field to zero. The receiver shall ignore this field.

9.2.12 PACKET SEQUENCE NUMBER (PSN) - 24 BITS

This field is used to identify the position of a packet within a sequence of packets. All IBA requesters shall generate a monotonically increasing (modulo 2^{24}) PSN when originating a packet. Depending upon the transport service type and / or implementation requirements, a responder may validate the PSN to detect missing packets.

9.3 EXTENDED TRANSPORT HEADERS

9.3.1 RELIABLE DATAGRAM EXTENDED TRANSPORT HEADER (RDETH) - 4 BYTES

Reliable Datagram Extended Transport Header (RDETH) contains the End-to-End Context identifier.

bits bytes	31-24	23-16	15-8	7-0
0-3	Reserve	EE-Context		

Figure 67 Reliable Datagram Extended Transport Header (RDETH)

9.3.1.1 RESERVE - 8 BITS

o9-2: If a CA implements Reliable Datagram functionality, then when generating a packet, the sender shall set this field to 0x0. The receiver shall ignore this field.

9.3.1.2 END-TO-END (EE) CONTEXT - 24 BITS

This field indicates the End-to-End (EE) Context used for this packet. EE context is a unique endnode identifier used to multiplex / demultiplex reliable datagram packets between any two end nodes. The EE-Context provides a context for reliable transfer state similar to that used for reliable connection.

9.3.2 DATAGRAM EXTENDED TRANSPORT HEADER (DETH) - 8 BYTES

Datagram Extended Transport Header (DETH) contains the additional transport fields for reliable and unreliable datagram service.

bits bytes	31-24	23-16	15-8	7-0
0-3	Queue Key			
4-7	Reserve	Source QP		

Figure 68 Datagram Extended Transport Header (DETH)

9.3.2.1 Q_KEY - 32 BITS

This field is required to authorize access to the destination queue. The responder compares this field with the destination's QP Q_Key.

9.3.2.2 RESERVE - 8 BITS

C9-8: When generating a packet, the sender shall set this field to 0x0. The receiver shall ignore this field.

9.3.2.3 SOURCE QP (SRCQP) - 24 BITS

This field specifies the source queue pair (QP) identifier. This is used as the destination QP for response packets.

9.3.3 RDMA EXTENDED TRANSPORT HEADER (RETH) - 16 BYTES

RDMA Extended Transport Header (RETH) contains the additional transport fields for RDMA operations.

bits bytes	31-24	23-16	15-8	7-0
0-3	Virtual Address (63-32)			
4-7	Virtual Address (31-0)			
8-11	R_Key			
12-15	DMA Length			

Figure 69 RDMA Extended Transport Header (RETH)

9.3.3.1 VIRTUAL ADDRESS (VA) - 64 BITS

Start address of buffer. RDMA VA may start on any byte boundary.

9.3.3.2 R_KEY - 32 BITS

R_Keys have the following properties:

- A R_Key acts as a protection key to access the specified memory address and range for a given operation, i.e. it is a protection mechanism to insure proper access to the target memory. The responder correlates the R_Key to the local protection mechanisms to validate the requester's access rights.
- A R_Key must be exported to the requester - this process (also includes the export of the starting virtual address and memory size, i.e. length) is outside the scope of this section.
- Access rights are granted for any combination of RDMA READ, RDMA WRITE, and ATOMICs - including none and all.
- Each Memory Region or Window has a single valid R_Key at any given moment. A virtually contiguous range of memory locations can have multiple Regions or Windows associated with it concurrently, each with an associated R_Key.
- A R_Key can be exported to multiple remote responders.
- R_Keys are used only for RDMA and ATOMIC Operations. A R_Key is contained within the packet header.

A responder that supports RDMA and / or ATOMIC Operations shall verify the R_Key, the associated access rights, and the specified virtual address. The responder must also perform bounds checking (i.e. verify that the length of the data being referenced does not cross the associated memory start and end addresses). Any violation must result in the packet being discarded and for reliable services, the generation of a NAK.

9.3.3.3 DMA LENGTH (DMALEN) - 32 BITS

This field indicates the length, in bytes, of the remote DMA operation.

C9-9: For an HCA performing RDMA operations, the minimum length specified in the DMALen field is 0; the maximum length is 2^{31}.

o9-3: If a TCA implements RDMA functionality, the minimum length specified in the DMALen field is 0; the maximum length is 2^{31}.

9.3.4 ATOMIC EXTENDED TRANSPORT HEADER (ATOMICETH) - 28 BYTES

ATOMIC Extended Transport Header (AtomicETH) contains the additional transport fields for ATOMIC Request operations.

bits bytes	31-24	23-16	15-8	7-0
0-3	Virtual Address (63-32)			
4-7	Virtual Address (31-0)			
8-11	R_Key			
12-15	Swap (or Add) Data (63-32)			
16-19	Swap (or Add) Data (31-0)			
20-23	Compare Data (63-32)			
24-27	Compare Data (31-0)			

Figure 70 ATOMIC Extended Transport Header (AtomicETH)

9.3.4.1 VIRTUAL ADDRESS (VA) - 64 BITS

Start address of buffer.

9.3.4.2 R_KEY - 32 BITS

R_Key used to verify remote access to the specified virtual address. See 9.3.3.2 R_Key - 32 bits on page 241.

9.3.4.3 SWAP (ADD) DATA (SWAPDT) - 64 BITS

The data operand used in ATOMIC Operations. In a CmpSwap operation this field is swapped into the addressed buffer if the CmpDt matched the existing buffer contents. In a FetchAdd operation this field is added to the contents of the addressed buffer.

9.3.4.4 COMPARE DATA (CMPDT) - 64 BITS

The data operand used in Compare portion of the CmpSwap ATOMIC Operation.

9.3.5 ACK EXTENDED TRANSPORT HEADER (AETH) - 4 BYTES

ACK Extended Transport Header (AETH) contains the additional transport fields for ACK packets. The ACK Extended Transport header is included in all ACK and the first and last packet of RDMA READ Response messages.

bits bytes	31-24	23-16	15-8	7-0
0-3	Syndrome	MSN		

Figure 71 Acknowledge Extended Transport Header (AETH)

9.3.5.1 SYNDROME

This field indicates if this is an ACK or NAK. If the packet is an ACK and the QP is associated with Reliable Connection transport service, the syndrome also provides the Limit Sequence Number (LSN) - see 9.7.7.2 End-to-End (Message Level) Flow Control on page 347. If packet is a NAK, it indicates the error code. For RNR NAK, this field indicates the responder's requested timer to be used before retransmitting the request.

9.3.5.2 MESSAGE SEQUENCE NUMBER (MSN)

Monotonically increasing (modulo 2^{24}) sequence number of the last message completed at the responder. This field is used to optimize completion processing at the requester.

9.3.5.3 ATOMIC ACKNOWLEDGE EXTENDED TRANSPORT HEADER (ATOMICACKETH) - 8 BYTES

ATOMIC Acknowledge Extended Transport Header (AtomicETH) contains the additional transport fields for ATOMIC response operations.

bits bytes	31-24	23-16	15-8	7-0
0-3	Original Remote Data (63-32)			
4-7	Original Remote Data (31-0)			

Figure 72 ATOMIC Acknowledge Extended Transport Header (AtomicAckETH)

9.3.5.4 ORIGINAL REMOTE DATA (ORIGREMDT) - 64 BITS

The data result from an ATOMIC Operation. This is the initial contents read from the remote memory buffer.

9.3.6 IMMEDIATE EXTENDED TRANSPORT HEADER (IMMDT) - 4 BYTES

Immediate Data (ImmDt) contains data that is placed in the receive Completion Queue Element (CQE). The ImmDt is only allowed in SEND or RDMA WRITE packets with Immediate Data.

bits bytes	31-24	23-16	15-8	7-0
0-3	Immediate Data			

Figure 73 Immediate Extended Transport Header (ImmDt)

9.3.7 INVALIDATE EXTENDED TRANSPORT HEADER (IETH) - 4 BYTES

The SEND with Invalidate operation carries with it an R_Key field. This R_Key is used by the responder to invalidate a memory region or memory window once it receives and executes the SEND with Invalidate request. The R_Key is carried in a new extended transport header called the Invalidate Extended Transport Header (IETH) as shown below.

bits bytes	31-24	23-16	15-8	7-0
0-3	R_Key			

Figure 74 Invalidate Extended Transport Header (IETH)

9.3.7.1 R_KEY - 32 BITS

R_Key defining the memory region or memory window to be Invalidated by the responder.

9.4 TRANSPORT FUNCTIONS

A QP provides the transport layer's client (e.g. the verbs layer in an HCA) with a specific transport service. Different transport services have various reliability levels for connected and connectionless communication. This section describes the basic functions used with each of the transport services. Additional transport sections go into more depth on the specifics of response packets, ordering, error recovery, etc. This section provides the high level view of the functions and how they work.

Not all the functions are available for each transport service, as described in Table 36 below. The Raw Datagram transport service does not use the

Table 36 Transport Functions Supported for Specific Transport Services

Transport Function	Transport Service				
	Reliable Connection	Unreliable Connection	Reliable Datagram	Unreliable Datagram	Raw Datagram
SEND	supported	supported	supported	supported	not applicable
RESYNC	not supported	not supported	supported	not supported	not supported
RDMA WRITE	supported	supported	supported	not supported	not applicable
RDMA READ	supported	not supported	supported	not supported	not applicable
ATOMIC Operations	optional support	not supported	optional support	not supported	not applicable

IBA defined transport functions. Instead, Raw Datagram packets transfer data that is part of some other, non IBA protocol.

9.4.1 SEND OPERATION

The SEND Operation is sometimes referred to as a Push operation or as having channel semantics. Both terms refer to how the SW client of the transport service views the movement of data. With a SEND operation the initiator of the data transfer pushes data to the remote QP. The initiator doesn't know where the data is going on the remote node. The remote node's Channel Adapter places the data into the next available receive buffer for that QP. On an HCA, the receive buffer is pointed to by the WQE at the head of the QP's receive queue.

The SEND Operation is referred to as having channel semantics because it moves data much like a mainframe IO channel -- the data is tagged with a discriminator (for IBA the discriminator is the destination LID and QP number) and the destination chooses where to place the data based on the discriminator.

A SEND Operation moves a single message. For the RC, RD, and UC transport services this message may be longer than a single packet. A message may range in size from zero bytes to 2^{31} bytes.

C9-10: The size of a SEND Operation, as generated by a requester, shall be between zero and 2^{31} bytes (inclusive).

C9-11: For RC and UC transport services in an HCA, a request message greater than PMTU in length shall be segmented into PMTU-sized segments for transmission via multiple packets. Similarly, an HCA responder shall reassemble such packets back into a single message.

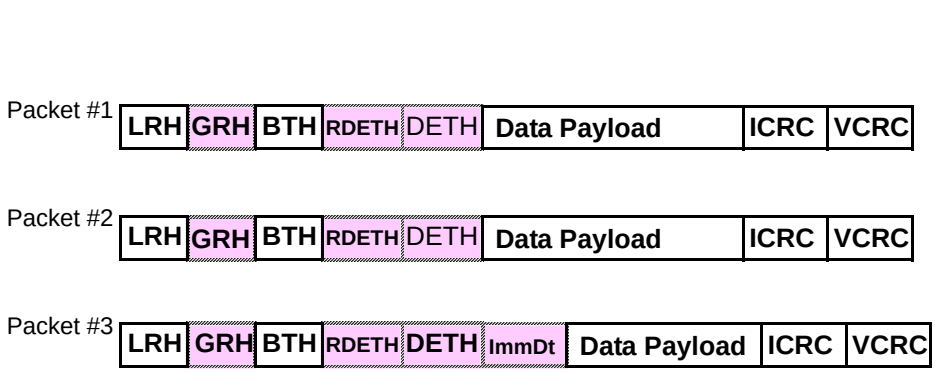
o9-4: For RD transport services in an HCA, a request message greater than PMTU in length shall be segmented into PMTU-sized segments for transmission via multiple packets. Similarly, an HCA responder shall reassemble such packets back into a single message.

o9-5: For RC, UC and RD transport services in a TCA, a request message greater than PMTU in length shall be segmented into PMTU-sized segments for transmission via multiple packets. Similarly, a TCA responder shall reassemble such packets back into a single message.

C9-12: For the Unreliable Datagram transport service, a SEND Operation shall consist only of single packet messages (i.e. the message data payload is limited to a maximum of the PMTU between the requester and the responder, i.e. 256, 512, 1024, 2048, or 4096 bytes).

A SEND Operation can, at the discretion of the client, include 4 bytes of Immediate data with each send message. If included, the Immediate data is contained within an additional header field (Immediate Extended Transport Header or ImmDt) on the last packet of the SEND Operation.

For example, Figure 75 below shows a SEND Operation of 700 bytes requiring 3 SEND packets, (assuming a 256 Byte PMTU).



Packet	BTH OpCode[a]
#1	"SEND First"
#2	"SEND Middle"
#3	"SEND Last" or "SEND Last with Immediate"

a. The BTH OpCode determines if the RDETH, DETH, and **ImmDt** headers are present.

A 700 byte SEND Operation uses 3 packets, assuming a 256 Byte PMTU. Acknowledgment Packets, used for reliable transport services, are not shown.

	Field Name
LRH	Local Route Header
GRH	Global Route Header
BTH	Base Transport Header
RDETH	Reliable Datagram Extended Transport Header[a]
DETH	Datagram Extended Transport Header[b]
ImmDt	Immediate Extended Transport Header
ICRC	Invariant CRC
VCRC	Variant CRC

a. Present only for the Reliable Datagram transport service
b. Present only for Reliable Datagram and Unreliable Datagram transport service

Figure 75 SEND Operation Example

There are several things to note from the above figure:

- The BTH OpCode field determines the start and end of the SEND message.
  - If the SEND message is less than or equal to the PMTU, then the BTH OpCode "SEND Only" or "SEND Only with Immediate" is used.
  - If the SEND message is for a length of zero, then the BTH OpCode "SEND Only" or "SEND Only with Immediate" is used. In this case, there is no Data Payload field, but all other fields are as shown.
  - If the SEND message is greater than the PMTU, then the BTH OpCode of the first packet is "SEND First" and the BTH OpCode of the last packet is "SEND Last" or "SEND Last with Immediate".
  - If the SEND message is greater than twice the PMTU, then the packets between the first and last use the BTH OpCode "SEND Middle".

- Every packet in a message that doesn't have the opcode SEND Only, SEND Only with Immediate, SEND Last, or SEND Last with Immediate shall have a data field of PMTU length.

- The responder node (the destination of the SEND Operation) does not know the final length of the SEND message until the last packet with the "SEND Last" or "SEND Last with Immediate" Op-Code arrives.

- The Packet Sequence Number field is used by the responder to detect out-of-order or missing packets.

- If the entire message is not a multiple of the PMTU, then the initial packets of the message carry a full PMTU number of bytes and the final packet carries the remainder as a partial payload.

- For a given requesting node's QP, once a multi-packet SEND Operation is started, no other request packets may be generated until the "SEND Last" or "SEND Last with Immediate" packet.

C9-13: A multi-packet message shall not be interleaved with other operations on the same SEND Queue.

- Not all SEND messages carry Immediate data. If they do, a special header is included in the last or only packet of the message. The presence of the header is indicated by a special "SEND Last with Immediate Data" or "SEND Only with Immediate Data" Op-Code in the BTH.

- For an HCA, there is no alignment requirement for the source or destination buffers of a SEND message. For buffers within a TCA, any alignment requirement is implementation specific.

The verbs chapter explains how the upper level SW client of an HCA uses a work request to post a buffer that is in turn segmented and sent as packets across the fabric. The same chapter also describes how the destination node posts a receive buffer into which the destination HCA reassembles the data. SEND messages initiated by a TCA use an implementation specific mechanism to create (and respond to) SEND packets.

C9-14: When generating a packet for a SEND operation, the requester shall include at least these headers and fields in every packet of the request: LRH, BTH, Data Payload, ICRC, VCRC.

C9-15: When generating a response to a SEND operation, the responder shall include at least these headers and fields the response: LRH, BTH, AETH, ICRC, VCRC.

9.4.1.1 SEND WITH INVALIDATE

In most respects, a SEND with Invalidate operates exactly as does a normal SEND operation. The significant variation is that the last (or only) packet of the message carries with it an R_KEY in an IETH (Invalidate Extended Transport Header). This is the R_KEY that the responder is being asked to invalidate. In effect, the Invalidate function is piggybacked onto a normal SEND operation.

There are several things to note about SEND with Invalidate with respect to a normal SEND operation. All of the following statements assume that an Invalidate operation is being piggybacked onto a “normal” SEND operation.

- If the SEND message onto which the invalidate operation is piggybacked is less than or equal to the PMTU, then the BTH OpCode “SEND Only with Invalidate” is used.
- If the SEND message onto which the invalidate operation is piggybacked is for a length of zero, then the BTH OpCode “SEND Only with Invalidate” is used. In this case, there is no Data Payload field, but all other fields are as shown.
- If the SEND message onto which the invalidate operation is piggybacked is greater than the PMTU, then the BTH OpCode of the first packet is “SEND First” and the BTH OpCode of the last packet is “SEND Last with Invalidate”. The BTH OpCode for all other packets of the message is “SEND Middle”.
- Every packet in a SEND message that doesn’t have the opcode “SEND Only”, “SEND Only with Immediate”, “SEND Only with Invalidate”, “SEND Last”, “SEND Last with Immediate”, or “SEND Last with Invalidate” shall have a data field of PMTU length.
- The responder node (the destination of the SEND operation) does not know the final length of the SEND message onto which the invalidate operation is being piggybacked until the last packet with the “SEND Last with Invalidate” OpCode arrives.
- For a given requesting node’s QP, once a multi-packet SEND operation is started, no other request packets may be generated until the “SEND Last”, “SEND Last with Immediate”, or “SEND Last with Invalidate” packet is generated.

Remote invalidate operations are used to disable access to memory through the R_Key contained in the IETH. However, Invalidate operations maintain the memory translation and protection resources associated with the R_Key that is being invalidated. A SEND with Invalidate operation does not disable all accesses to the memory referenced by the R_Key, it simply means that the R_Key being invalidated can no longer be used to access those memory regions or memory windows. Other R_Keys that reference the same memory remain valid.

The intent is that the Invalidate operation be executed by the channel interface, which may include hardware as well as software drivers below the verbs. Thus, the invalidate operation may be performed by the software driver immediately above the transport layer hardware. In this case, the transport layer hardware simply passes the IETH upwards to the software portion of the channel interface which executes the necessary invalidation. For this reason, the IETH is not validated as part of normal transport packet header validation. The invalidate operation does not make any changes to the state of the referenced memory.

9.4.1.1.1 INVALIDATE OPERATION ORDERING

o9-5.2.1: For any HCA which supports SEND with Invalidate, upon receiving an IETH, the Invalidate operation must not take place until after the normal transport header validation checks have been successfully completed.

An invalidation must not occur if the packet containing the IETH (e.g. SEND with Invalidate) is not a valid packet as described by normal packet validation procedures. However, since the invalidation operation is not executed by the transport layer, the Invalidate operation may take place either before or after the transport-level acknowledge has been generated, but in any case, the transport-level acknowledgement (ACK/NAK) does not reflect either the success or failure of the invalidate operation.

The following defines the ordering rules for a SEND with Invalidate operation:

1) there are no ordering guarantees between any SEND with Invalidate operation and any subsequent operation. Thus a requester cannot rely on a SEND with Invalidate operation, by itself, to prevent access to the responder's invalidated region by a subsequent operation.
2) Normal ordering rules apply to an RDMA WRITE, SEND or ATOMIC operation followed by a SEND with Invalidate operation. In this case, the RDMA WRITE, SEND or ATOMIC operation executes before the subsequent SEND with Invalidate operation is executed at the responder.
3) a SEND with Invalidate operation may impact a previous RDMA READ operation. Thus, a requester should not perform a SEND with Invalidate while previous RDMA READ operations are still outstanding. The requester can set the Fence attribute on a given work request such as a SEND with Invalidate in order to ensure that previous outstanding RDMA READ operations have completed before initiating a subsequent SEND with Invalidate operation.
4) As always, acknowledgements are always returned by the responder in order, and WQEs at the requester are always completed in order.

9.4.1.1.2 RESPONDER - R_KEY VALIDATION

The transport layer does not validate the R_Key field of the IETH during transport-level packet header validation. The channel interface retains responsibility for validating the R_Key field of the IETH prior to executing the invalidation. For this reason, the IETH is not considered to be a transport header and therefore R_Key validation for a remote invalidate operation is not discussed in this section.

See 9.4.1.1.3 R_Key Validation for Remote Memory Invalidate on page 251 for a description of validating the R_Key field of the IETH.

9.4.1.1.3 R_KEY VALIDATION FOR REMOTE MEMORY INVALIDATE

o9-5.2.2: For a CA which supports remote invalidate, prior to executing a remote invalidate operation, the R_Key contained in the IETH must be validated. To be considered valid, all of the following conditions must be true:

1) The R_Key must not be in the invalid state.
2) If the implementation supports a single key space for the L_Keys and R_Keys, the R_Key contained in the IETH must not be the same as the reserved L_Key.
3) For an HCA, the R_Key must not reference a memory region that was created through a Register Memory Region or Reregister Memory Region.
4) For an HCA, the R_Key must not reference a Shared Memory Region.
5) For an HCA, the R_Key must not reference a memory region that is associated with a different protection domain than the QP on which the IETH was received.
6) For an HCA, the R_Key must not reference a type 1 memory window. See Section 10.6.7.2.3 Type 1 Memory Windows on page 495 for a description of type 1 memory windows.
7) For an HCA, the R_Key must not simultaneously be in the valid state and be referencing a type 2A memory window that is associated with a different QP than the QP on which the IETH was received. See Section 10.6.7.2.4 Type 2 Memory Windows on page 497 for a description of type 2 memory windows.
8) For an HCA, the R_Key must not simultaneously be in the valid state and be referencing a type 2B memory window that is associated with a different protection domain or QP than the QP on which the IETH was received. See Section 10.6.7.2.4 Type 2 Memory Windows on page 497 for a description of type 2 memory windows.
9) For an HCA, the R_Key must not simultaneously be in the free state and be referencing a type 2 memory window that is associated with a

different protection domain than the protection domain of the QP on which the IETH was received. See Section 10.6.7.2.4 Type 2 Memory Windows on page 497 for a description of type 2 memory windows.

Note: If the L_Key and its accompanying R_Key are in the Free State then a remote invalidate operation will not change the L_Key and its accompanying R_Key state.

The behavior governing the QP if the R_Key validation as defined above in compliance statement o9-5.2.2: fails is defined in Table 58 Responder Error Behavior Summary on page 409 under the entry titled “Remote Invalidate Error”.

9.4.2 RESYNC OPERATION

A RESYNC operation is supported only for the Reliable Datagram transport service. RESYNC is essentially the same as a zero-length Reliable Datagram SEND-only request but with a several unique properties:

1) RESYNC is used by the requester to force the responder to reset its expected PSN to a value defined by the requester,
2) A RESYNC request carries a data payload of zero length,
3) The responder is required to accept a RESYNC request, even if the currently executing request has not yet completed.
4) A RESYNC request does not, itself, directly consume either a send WQE on the requester side, nor a receive WQE on the responder side.

C9-15.a1: The RESYNC request shall carry a zero length Data Payload.

9.4.3 RDMA WRITE OPERATION

The RDMA WRITE Operation is used by the requesting node to write into the virtual address space of a destination node. The message may be between zero and 2^{31} bytes (inclusive) and is written to a contiguous range of the destination QP’s virtual address space (not necessarily a contiguous range of physical memory).

C9-16: For an HCA requester performing RDMA WRITE operations, the length of an RDMA WRITE message, as reflected in the RETH:DMALen field, shall be between zero and 2^{31} bytes (inclusive).

o9-6: If a TCA requester implements RDMA WRITE functionality, the length of an RDMA WRITE message, as reflected in the RETH:DMALen field, shall be between zero and 2^{31} bytes (inclusive).

Before allowing incoming RDMA WRITEs, the destination node first allocates a memory range for access by the destination’s QP (or group of

QPs). A destination's channel adapter associates a 32-bit R_Key with this memory region or window. For a HCA, the verbs layer refers to this as registering a memory region - see 10.6 Memory Management on page 468. TCAs use an implementation-specific mechanism to allocate and manage R_Keys that is outside the scope of the IBA specification.

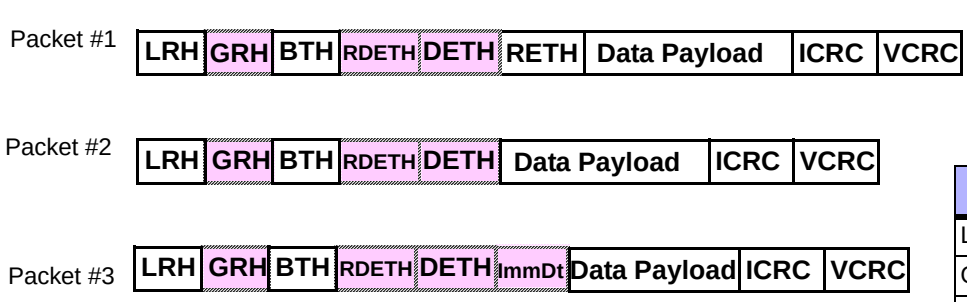
The destination communicates the virtual address, length, and R_Key to any other host it wishes to have access the memory region. The communication of address and R_Key is done by the client upper level protocol - the exchange is outside the scope of the IBA. For example, an application program might embed the address, length, and R_Key into a private data structure that it in turn pushes to other application programs using the SEND Operation.

C9-17: As with SEND Operations, an HCA requester shall segment a RDMA WRITE message larger than the PMTU into multiple packets.

o9-7: If a TCA requester implements RDMA WRITE functionality, it shall segment a RDMA WRITE message larger than the PMTU into multiple packets.

If specified by the verbs layer, Immediate data is included in the last packet of an RDMA WRITE message. The Immediate data is not written to the target virtual address range, but is passed to the client after the last RDMA WRITE packet is successfully processed. E.G. on an HCA the immediate data is placed on the completion queue.

For example, Figure 76 below shows a 700 byte RDMA WRITE (on a path with a 256B PMTU).



Packet	BTH OpCode[a]
#1	"RDMA WRITE First"
#2	"RDMA WRITE Middle"
#3	"RDMA WRITE Last" or "RDMA WRITE Last with Immediate"

a. The BTH OpCode field determines if the RDETH, DETH, and ImmDt headers are present.

A 700 byte RDMA WRITE Operation uses 3 packets, assuming a 256 Byte PMTU. Acknowledgment Packets, used for reliable transport services, are not shown.

	Field Name
LRH	Local Route Header
GRH	Global Route Header
BTH	Base Transport Header
RDETH	Reliable Datagram Extended Transport Header[a]
DETH	Datagram Extended Transport Header[a]
RETH	RDMA Extended Transport Header
ImmDt	Immediate Extended Transport Header
ICRC	Invariant CRC
VCRC	Variant CRC

a. Present only for the Reliable Datagram transport service

Figure 76 RDMA WRITE Operation Example

There are several things to note from the above figure:

- The BTH OpCode field determines the start and end of the RDMA WRITE message.
  - If the RDMA WRITE request was for a length of zero, then the BTH OpCode "RDMA WRITE Only" or "RDMA WRITE Only with Immediate" is used. In this case, there is no Data Payload field, but all other fields are as shown.
  - If the RDMA WRITE message is less than or equal to the PMTU, then the BTH OpCode "RDMA WRITE Only" or "RDMA WRITE Only with Immediate" is used.
  - If the RDMA WRITE message is greater than the PMTU, then the BTH OpCode of the first packet is "RDMA WRITE First" and the BTH OpCode of the last packet is "RDMA WRITE Last" or "RDMA WRITE Last with Immediate".

- If the RDMA WRITE message is greater than twice the PMTU, then the packets between the first and last use the BTH OpCode “RDMA WRITE Middle”.
- Every packet in a RDMA WRITE message that doesn’t have the opcode RDMA WRITE Only, RDMA WRITE Only with Immediate, RDMA WRITE Last, or RDMA WRITE Last with Immediate has a data field of PMTU length.

- The RETH header is present in the first (or only) packet of the message. It contains the virtual address of the destination buffer as well as the R_Key and message length fields.
- The Packet Sequence Number field is used by the responder to detect out-of-order or missing packets.
- If the entire message is not a multiple of the PMTU, then the initial packets of the message carry a full PMTU number of bytes and the final packet carries the remainder in a partial payload.
- For a given requesting node’s QP, once a multi-packet RDMA WRITE operation is started, no other request packets may be generated until the “RDMA Last” or “RDMA Last with Immediate Data” packet is sent.

C9-18: For an HCA RDMA WRITE request, a multi-packet message shall not be interleaved with other operations on the same SEND Queue.

o9-8: If a TCA requester implements RDMA WRITE functionality, then for an RDMA WRITE request, a multi-packet message shall not be interleaved with other operations on the same SEND Queue.

- Not all RDMA WRITE messages carry Immediate data. If a RDMA WRITE does, a special header is included in the last (or only) packet of the message. The presence of the header is indicated by a special “RDMA WRITE Last with Immediate Data” or “RDMA WRITE Only with Immediate Data” OpCode in the BTH.
- For an HCA, there is no alignment requirement for the source or destination buffers of an RDMA WRITE message. For buffers within a TCA, any alignment requirement is implementation specific.

C9-19: When generating an RDMA WRITE Request, an HCA requester shall include at least the following headers and fields in each request packet: LRH, BTH, Data Payload, ICRC, VCRC. The first (or only) packet of the request shall also include the RETH.

o9-9: If a TCA requester implements RDMA WRITE functionality, it shall behave as follows. When generating an RDMA WRITE Request, a TCA requester shall include at least the following headers and fields in each request packet: LRH, BTH, Data Payload, ICRC, VCRC. The first (or only) packet of the request shall also include the RETH.

C9-20: When generating an RDMA WRITE Response, an HCA responder shall include at least the following headers and fields in each response packet: LRH, BTH, AETH, ICRC, VCRC.

o9-10: If a TCA responder implements RDMA WRITE functionality, then when generating an RDMA WRITE Response, a TCA responder shall include at least the following headers and fields in each response packet: LRH, BTH, AETH, ICRC, VCRC.

9.4.4 RDMA READ OPERATION

RDMA READ Operations are similar to RDMA WRITE Operations. They allow the requesting node to read a virtually contiguous block of memory on a remote node. As with RDMA WRITEs, the responding node first allows the requesting node permission to access its memory. The responder passes to the requestor a virtual address, length, and R_Key to use in the RDMA READ request packet.

A single RDMA READ request can read from zero to 2^{31} bytes (inclusive) of data.

C9-21: For an HCA responding to an RDMA READ request, if the requested data size is greater than the PMTU, the responder shall segment the data into PMTU size data segments for transmission as multiple RDMA READ Response packets. The data is reassembled in the requesting node's memory.

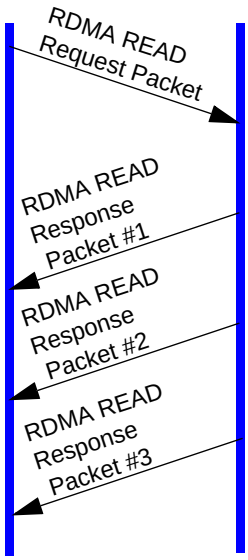
o9-11: If a TCA responder implements RDMA READ functionality, and the requested data size is greater than the PMTU, the responder shall segment the data into PMTU size data segments for transmission as multiple RDMA READ Response packets. The data is reassembled in the requesting node's memory.

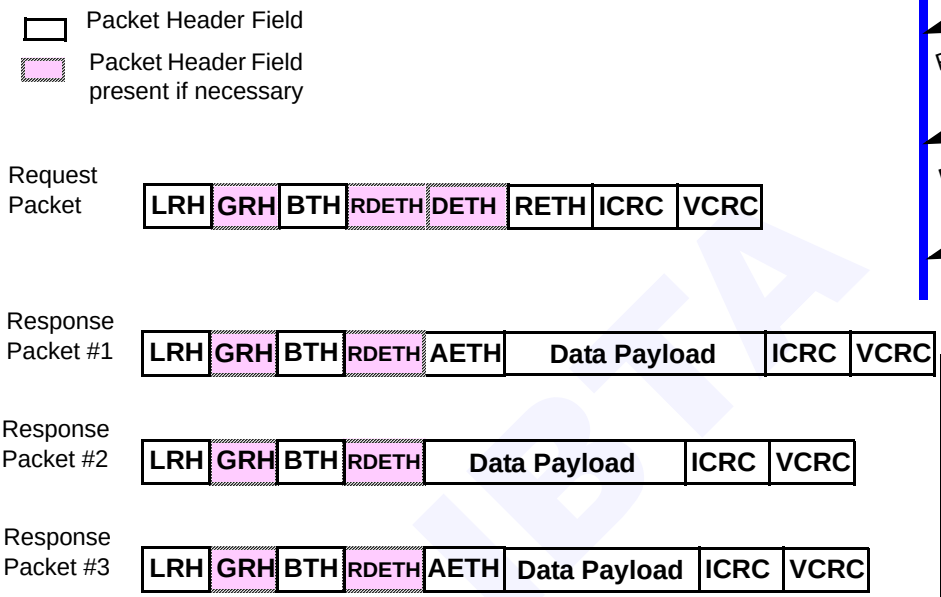
C9-22: For an HCA requester using RDMA operations, the length of the requested RDMA READ data, as reflected in the RETH:DMALen field, shall be between zero and 2^{31} bytes (inclusive).

o9-12: If a TCA requester implements RDMA READ functionality, then the length of the requested RDMA READ data, as reflected in the RETH:DMALen field, shall be between zero and 2^{31} bytes (inclusive).

The following example in Figure 77 shows a 700 byte RDMA READ operation (on a path with a 256B PMTU)

A ladder diagram showing the single RDMA READ Request Packet initiated by the requestor node. In this example, the destination node segments the data into three response packets.





Packet	BTH OpCode[a]
Request	"RDMA READ Request"
#1	"RDMA READ Response First"
#2	"RDMA READ Response Middle"
#3	"RDMA READ Response Last"

a. The BTH OpCode field determines if the RDETH and AETH are present.

A 700 byte RDMA READ Operation has 3 response packets, assuming a 256 Byte PMTU.

	Field Name
LRH	Local Route Header
GRH	Global Route Header
BTH	Base Transport Header
RDETH	Reliable Datagram Extended Transport Header[a]
DETH	Datagram Extended Transport Header[a]
AETH	Acknowledgment Extended Transport Header
RETH	RDMA Extended Transport Header
ICRC	Invariant CRC
VCRC	Variant CRC

a. Present only for the Reliable Datagram transport service

Figure 77 RDMA READ Operation Example

There are several items to note in the previous figure:

- A single request packet will result in multiple read response packets if the read length is greater than the PMTU.
- The BTH OpCode field identifies the packet as a RDMA READ Request or Response as well as determines if any of the extended transport headers are present.
- The BTH OpCode field determines the start and end of the RDMA READ Acknowledgment message.
  - If the RDMA READ request message requested a zero byte transfer, then the BTH OpCode “RDMA READ Response Only” is used. All other fields remain as shown.
  - If the RDMA READ Acknowledgment message is less than or equal to the PMTU, then the BTH OpCode “RDMA READ Response Only” is used.
  - If the RDMA READ message is greater than the PMTU, then the BTH OpCode of the first packet is “RDMA READ Response First” and of the last packet “RDMA READ Response Last”.
  - If the RDMA READ response message is greater than twice the PMTU, then the packets between the first and last use the BTH OpCode “RDMA READ Response Middle”.
  - Every packet in a RDMA READ Response First or RDMA READ Response Middle message has a data field of PMTU length.
- If the entire message is greater than a multiple of the PMTU, then the initial packets of the response message carry a full PMTU number of bytes and the final packet carries a partial payload.
- The Packet Sequence Number (PSN) field is used to detect out-of-order or missing response packets.
- After initiating a RDMA READ Request packet, the requesting node may send out additional request packets without waiting for the response packets to return. See section 9.7.3.1 Requester Side - Generating PSN on page 289 for an explanation of how the PSN is determined for subsequent request packets.
- The maximum number of RDMA READ Requests for a particular QP that can be outstanding at any one time is negotiated at connection establishment time. A responder may restrict the connection to as few as one outstanding RDMA READ request per QP. If ATOMIC Operations are supported, the number of outstanding requests negotiated at connection establishment time includes both ATOMIC Operation requests and RDMA READ requests.
- RDMA READ packets never carry Immediate data.

RDMA READ Requests are retried if the requester did not receive the proper response.

- Retried RDMA READ Requests need not start at the same address nor have the same length as the original RDMA READ. The retried request may only reread those portions that were not successfully responded to the first time.
- The responder validates the R_Key and RDMA READ virtual address for the retried request.
- The PSN of the retried RDMA READ must be in the duplicate PSN region. See Section 9.7.1 Packet Sequence Numbers (PSN) on page 282
- The PSN of the retried RDMA READ request need not be the same as the PSN of the original RDMA READ request. Any retried request must correspond exactly to a subset of the original RDMA READ request in such a manner that all potential duplicate response packets must have identical payload data and PSNs regardless of whether it is a response to the original request or a retried request.
- For an HCA, there is no alignment requirement for the source or destination buffers of an RDMA READ message. For buffers within a TCA, any alignment requirement is implementation specific.

C9-23: When generating an RDMA READ Request, an HCA requester shall include at least the following headers and fields in its request packet: LRH, BTH, RETH, ICRC, VCRC.

o9-13: If a TCA requester implements RDMA operations, then it shall include at least the following headers and fields in its request packet: LRH, BTH, RETH, ICRC, VCRC.

C9-24: When generating an RDMA READ Response, an HCA responder shall include at least the following headers and fields in each response packet: LRH, BTH, Data Payload, ICRC, VCRC. If the response packet BTH:Opcode is “RDMA READ Response First, RDMA READ Response Last, or RDMA READ Response Only, the packet shall also include an AETH. If the response packet BTH:Opcode is “RDMA READ Response Middle, an AETH shall not be included.

o9-14: If a TCA responder implements RDMA operations, then it shall include at least the following headers and fields in each response packet: LRH, BTH, Data Payload, ICRC, VCRC. If the response packet BTH:Opcode is “RDMA READ Response First, RDMA READ Response Last, or RDMA READ Response Only, the packet shall also include an AETH. If the response packet BTH:Opcode is “RDMA READ Response Middle, an AETH shall not be included.

9.4.5 ATOMIC OPERATIONS

ATOMIC Operations execute a 64-bit operation at a specified address on a remote node. The operations atomically read, modify and write the destination address and guarantee that operations on this address by other QPs on the same CA do not occur between the read and the write. The scope of the atomicity guarantee may optionally extend to other CPUs and HCAs.

ATOMIC Operations use the same remote memory addressing mechanism as RDMA READs and Writes. The virtual address specified in the request packet is in the address space of the remote QP that the ATOMIC Operation has targeted.

ATOMIC Operations consist of two packet types, the “ATOMIC Command”, request packet and the “ATOMIC Acknowledge” response packet.

1) ATOMIC Operations are only supported by the Reliable Connection and Reliable Datagram transport services.
2) ATOMIC Operations do not support Immediate data.
3) ATOMIC Operations support is strongly recommended to be provided strictly in hardware.[7]
4) The virtual address in the ATOMIC Command Request packet shall be naturally aligned to an 8 byte boundary. The responding CA checks this and returns an Invalid Request NAK if it is not naturally aligned.

IBA defines the following ATOMIC Operations:

- **FetchAdd** (Fetch and Add)

  The FetchAdd ATOMIC Operation tells the responder to read a 64-bit buffer value at a naturally aligned virtual address in the responder's memory, perform an unsigned[8] add using the 64-bit Add Data field in the AtomicETH, and write the result (must match the memory type at the requester) back to the same virtual address. The responder's operation shall be atomic (i.e. undisturbed by other entities) per section 9.4.5.1 Atomicity Guarantees on page 262.

  The FetchAdd operation is performed in the endian format of the target memory. The original remote data is converted from the endian format of the target memory for return. The fields are in Big-endian format on the wire.

7. CA implementations may use software assists - this shall be indistinguishable from a hardware-only implementation; Performance must be such that higher level software applications are not affected.
8. If Signed numbers are used, this is the same as using twos complement arithmetic (the carry is not saved nor reported).

The requestor specifies:

- Remote data address and R_Key
- Add data

The acknowledge packet returns:

- Original remote data

After the operation, the responder's memory at the specified virtual address contains the unsigned sum of the original value and the Add field in the AtomicETH header. All operations on the requester's memory are done in the native endian format of the requester.

- **CmpSwap** (Compare and Swap)

  The CmpSwap ATOMIC Operation tells the responder to read a 64-bit value at a naturally aligned virtual address in the responder's memory, compare it with the Compare Data field in the AtomicETH header, and, if they are equal, write the Swap Data field from the AtomicETH header into the same virtual address. If they are not equal, the contents of the responder's memory are not changed. In either case, the original value read from the virtual address is returned to the requester. The responder's operation shall be atomic (i.e. undisturbed by other entities) per section 9.4.5.1 Atomicity Guarantees on page 262.

  The requestor specifies:

  - Remote data address and R_Key
  - Write (swap) data
  - Compare data

  The acknowledgment packet returns:

  - Original remote data

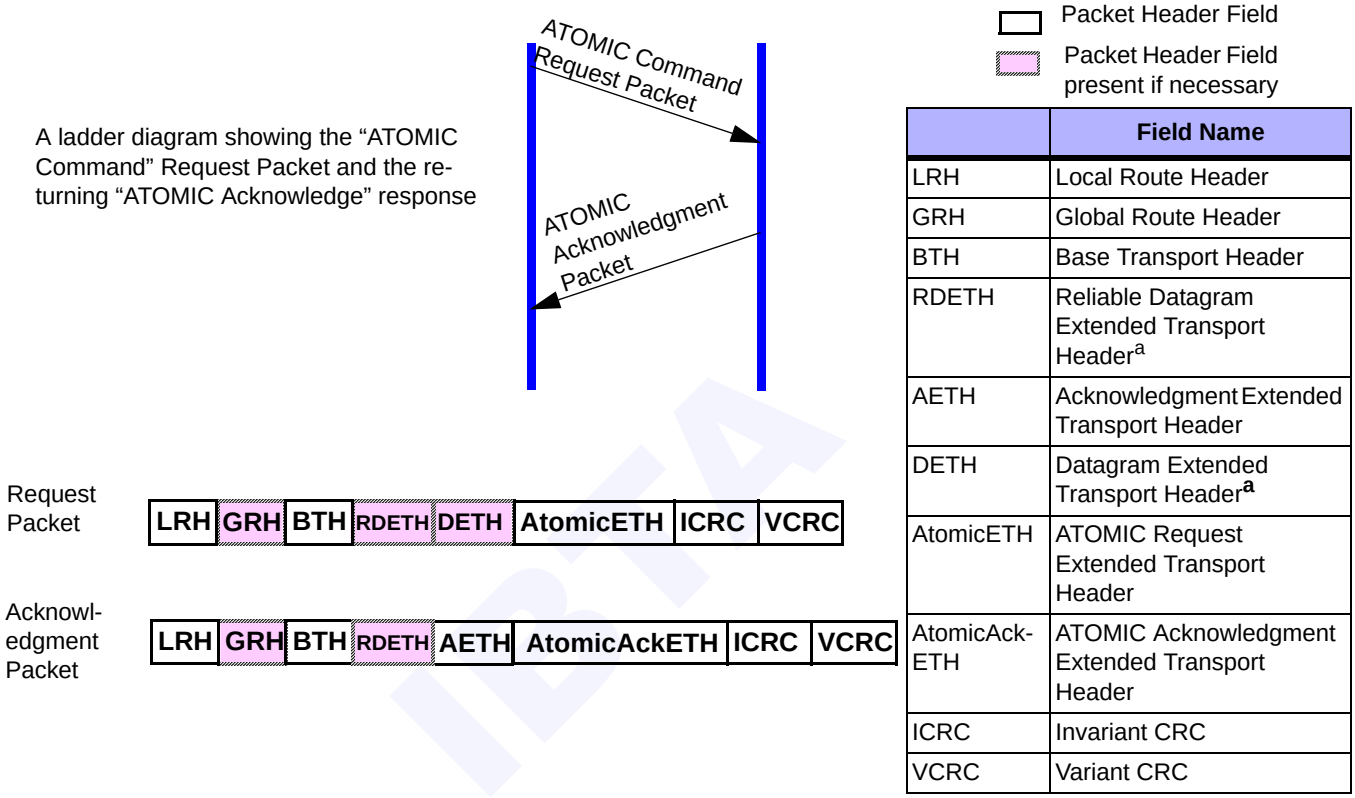
  After the operation, the remote data buffer contains the “original remote value” (if comparison did not match) or the “Write (swap) data” (if the comparison did match).

  The CmpSwap operation involves three 8 byte data buffers, the compare data, the write (swap) data, and the original remote data. All three are transmitted within the request and response packets in byte big endian format. All operations on the responder’s CA memory are done in the native endian format of that memory system. All operations on the requestor’s memory are done in the native endian format of the requestor.

  For example, consider a big endian CA initiating a CmpSwap ATOMIC Operation request packet to a little endian responder. The request packet contains two big endian data fields: the compare data and the write (swap) data. The responder converts these data fields to little endian format and does the compare and swap operation. The original

target data field is converted to big endian format and returned in the response packet.

.



	Field Name
LRH	Local Route Header
GRH	Global Route Header
BTH	Base Transport Header
RDETH	Reliable Datagram Extended Transport Header[a]
AETH	Acknowledgment Extended Transport Header
DETH	Datagram Extended Transport Header[a]
AtomicETH	ATOMIC Request Extended Transport Header
AtomicAck-ETH	ATOMIC Acknowledgment Extended Transport Header
ICRC	Invariant CRC
VCRC	Variant CRC

a. Present in ATOMIC Operations only for the Reliable Datagram transport service

Figure 78 ATOMIC Operation Example

o9-15: When generating an ATOMIC Operation request, a requester shall include at least an LRH, a BTH, an AtomicETH, an ICRC and a VCRC. The sources of data for the LRH, BTH and AtomicETH headers shall be as shown in Table 60 Packet Fields and Parameters by Service on page 420.

o9-16: When responding to an ATOMIC Operation request, a responder shall include in its response packet at least an LRH, BTH, AETH, AtomicAckETH, ICRC and a VCRC.

9.4.5.1 ATOMICITY GUARANTEES

o9-17: Atomicity of the read/modify/write on the responder’s node by the ATOMIC Operation shall be assured in the presence of concurrent atomic accesses by other QPs on the same CA.

o9-18: A CA may optionally assure atomicity of ATOMIC Operations in the presence of concurrent memory accesses from other CAs, IO devices, and CPUs. For a HCA, the Verbs layer shall report whether it supports this enhanced atomicity guarantee.

9.4.5.2 ATOMIC ACKNOWLEDGMENT GENERATION AND ORDERING RULES

1) For the requestor, an ATOMIC Operation is considered complete when the response packet returns.
2) If an RDMA READ work request is posted before an ATOMIC Operation work request then the atomic may execute its remote memory operations before the previous RDMA READ has read its data. This can occur because the responder is allowed to delay execution of the RDMA READ. Strict ordering can be assured by posting the ATOMIC Operation work request with the fence modifier. See the description for the fence modifier Post Send Request. The fence modifier causes the requestor to wait till the RDMA READ completes before issuing the ATOMIC Operation.
3) When a sequence of requests arrives at a QP, the ATOMIC Operation only accesses memory after prior (non-RDMA READ) requests access memory and before subsequent requests access memory. Since the responder takes time to issue the response to the atomic request, and this response takes more time to reach the requestor and even more time for the requestor to create a completion queue entry, requests after the atomic may access the responders memory before the requestor writes the completion queue entry for the ATOMIC Operation request.
4) Each ATOMIC Operation request requires an explicit response and acknowledge message. An ATOMIC Operation response, with a properly formed AETH, is considered an acknowledge message.

9.4.5.3 ERROR BEHAVIOR

A responder utilizes vendor specific resources and facilities to implement ATOMIC Operations and RDMA READs as well as to facilitate retried ATOMIC requests. It is the responsibility of the requestor to ensure that all unacknowledged ATOMIC operations and RDMA READs combined do not overrun the receiver resources. The number of these resources is negotiated on a per QP basis at connection setup (see 9.4.4 RDMA READ Operation on page 256 and 9.4.5 ATOMIC Operations on page 260).

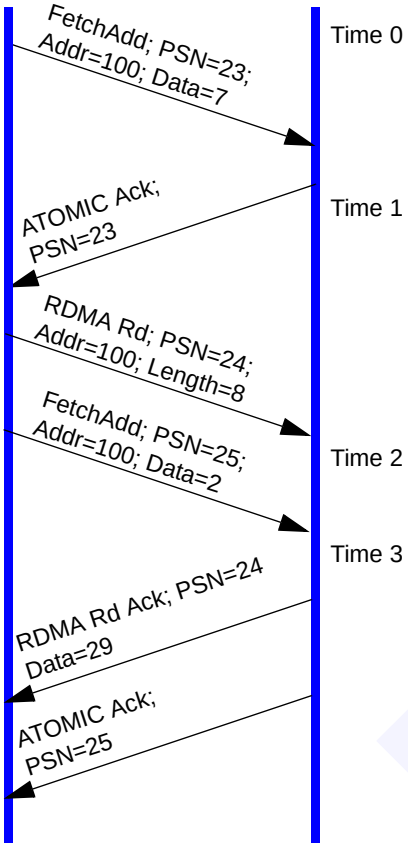
The responding node saves the reply data, the PSN, and an indication that the stored data is from an ATOMIC Operation. This saved data is used to generate the response for retried ATOMIC Operations. Note that the execution of an RDMA READ operation may consume the same resources as is used to save the ATOMIC Operation PSN and reply data. The information is stored in the destination QP's "connection context". The "connection context" is the QP context for Reliable Connection Ser-

vice. For Reliable Datagram Service, the “Connection context” is actually the “EE context”.

Several rules determine when the responder stores the PSN and reply data of an ATOMIC Operation:

- Only valid, new ATOMIC Operation requests (i.e. all header checks are valid, the incoming PSN matches the expected PSN, the R_Key is valid for the data being accessed, and the address is aligned to a 64b boundary) are saved.
- If the responder QP supports multiple outstanding ATOMIC Operations and RDMA READ Operations, the information on each valid request is saved in FIFO order. The FIFO depth is the same as the maximum number of outstanding ATOMIC Operations and RDMA READ requests negotiated on a per QP basis at connection setup.
- Repeated ATOMIC or RDMA READ Operations are not saved again.

The saved ATOMIC and RDMA READ state is shown in the figure below.



Timeline of Responder's State

Time	Contents of Memory at address 100	Per QP State[a] Tracking Most Recent ATOMIC & RDMA READ Operations					
		Most Recent			2nd Most Recent		
		Op	PSN	Result	Op	PSN	Result
Time 0	20	na	na	na	na	na	na
Time 1	27	ATOMIC	23	20	na	na	na
Time 2	27	RDMA READ	24	na	ATOMIC	23	20
Time 3	29	ATOMIC	25	27	RDMA READ	24	na

a. For Reliable Connection Service, the state bits for tracking the most recent ATOMIC and RDMA READ Operations are kept in the per QP State. For Reliable Datagram Service these state bits are kept in the EE Context instead of the Per QP State.

The ladder diagram shows multiple ATOMIC and RDMA READ Requests. In this example the responder's QP has agreed at connection setup time that it can accept up to any combination of 2 outstanding ATOMIC or RDMA READ Operations. This example shows how the responder maintains state of the recent ATOMIC and RDMA READ operations. Note also in the example a RDMA READ preceding an ATOMIC Operation but targeting the same address may return the value after the ATOMIC executes. Strict ordering is possible in an HCA by using the "fence" option when posting the ATOMIC following the RDMA READ. See 11.4.1.1 Post Send Request on page 612

Figure 79 Responder State Maintained for ATOMIC & RDMA READ Operations

An ATOMIC Operation is guaranteed to execute at most once. If the ATOMIC Operation does not execute on the destination, it is reported to the sender (e.g. an R_Key protection fault) with the appropriate NAK syndrome response.

However, like all operations, a non-recoverable error that occurs after execution at the responder, but before the response reaches the requester (e.g. a fatal HCA error), results in the requester not knowing the state of the responder's memory. This case must be detected and dealt with by the client or upper layer protocol.

As with all operations, errors could occur on any of the transfers. If the original "ATOMIC Command" request is lost, or the "ATOMIC Acknowledge" is lost, the sender will retry using the normal retry procedures. If the

retry fails, it is not certain whether the ATOMIC Operation took place at the destination, but the connection will be in the Error state.

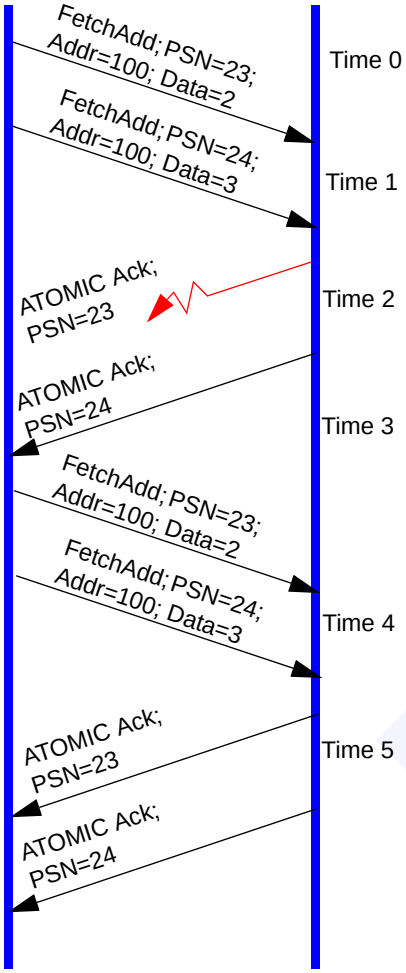
As with retries of Send and RDMA WRITE operations, if the responding CA has actually executed the request, it will only acknowledge the request again, not re-run the ATOMIC Operation. This is necessary since an ATOMIC Operation is not idempotent. The responder recognizes a retried ATOMIC Operation and returns the reply data from the original acknowledgment that was previously stored in the QP (or EE context for Reliable Datagram service) “hidden state”. The responder returns the stored result of an ATOMIC Operation if the following conditions are met:

- The request is valid (i.e. header an OpCode are valid)
- The request is for an ATOMIC Operation (the responder may check the ATOMIC Operation OpCode is the same as that of the stored operation)
- The PSN of the request is in the “duplicate region”. See a description of the PSN space in Section 9.7.1 Packet Sequence Numbers (PSN) on page 282.
- The PSN matches that of a saved ATOMIC Operation.

A retried ATOMIC Operation that does not meet the above conditions is discarded by the responder. See Table 58 Responder Error Behavior Summary on page 409

When an ATOMIC Operation is retried, the responder does not validate the R_Key nor does it translate the virtual address in the retried request.

The figure below demonstrates a failed ATOMIC Operation response packet and shows the retried request and the eventual successful response.



Timeline of Responder's State

Time	Contents of Memory at address 100	Per QP State[a] Tracking Most Recent ATOMIC & RDMA READ Operations					
		Most Recent			2nd Most Recent		
		Op	PSN	Result	Op	PSN	Result
Time 0	20	na	na	na	na	na	na
Time 1	22	ATOMIC	23	20	na	na	na
Time 2	25	ATOMIC	24	22	ATOMIC	23	20
Time 3	25	ATOMIC	24	22	ATOMIC	23	20
Time 4	25	ATOMIC	24	22	ATOMIC	23	20
Time 5	25	ATOMIC	24	22	ATOMIC	23	20

a. For Reliable Connection Service, the state bits for tracking the most recent ATOMIC and RDMA READ Operations are kept in the per QP State. For Reliable Datagram Service these state bits are kept in the EE Context instead of the Per QP State.

The ladder diagram shows multiple ATOMIC and RDMA READ Requests. In this example the responder's QP has agreed at connection setup time that it can accept up to any combination of 2 outstanding ATOMIC or RDMA READ Operations. This example shows a lost ATOMIC acknowledgment (at Time 2). When the request is retried, the original result value is returned. The original value is returned even if subsequent operations from the same or a different QP have modified the target of the ATOMIC Operation.

Figure 80 Retrying ATOMIC Operations

If all retries fail, that implies that the connection is lost, and the error recovery routines in the requesting CA's driver will inform the local application.

The size of the operation is always 64-bits. The target must be naturally aligned (low 3 bits of the virtual address must be zero). An error will be reported if the R_Key range does not fully enclose the target. If this or another protection error occurs, it will be reported (NAK_Remote_Access) but will not result in taking any of the "ATOMIC Operation hidden queue" resources. That is, if the same request is repeated (same PSN) and the responding side has subsequently allocated an R_Key range, this new operation will now succeed.

9.4.6 RESERVED AND MANUFACTURER DEFINED TRANSPORT FUNCTION OPCODES

The IBA has two mechanisms for future expansion of its transport layer:

- Reserved and Manufacturer Defined BTH OpCodes

IBA Transport layer functionality can be expanded by defining new BTH OpCodes. Two blocks of undefined OpCodes are specified. One for future revisions of the IBA and one block for manufacturer specific functions.

9.5 TRANSACTION ORDERING

This section defines the rules for ordering of transmission, execution, and completion for transactions for a given QP:

C9-25: A requester shall transmit request messages in the order that the Work Queue Elements (WQEs) were posted.

C9-26: For messages that are segmented into PMTU-sized packets, the data payload shall use the same order as the data segments defined by the WQE.

Packets from a given source QP to a given destination QP travel on the same path through the fabric and are received in the same order they were injected.

C9-27: For reliable services on an HCA, all acknowledge packets shall be strongly ordered, e.g. all previous RDMA READ responses and ATOMIC responses shall be injected into the fabric before subsequent SEND, RDMA WRITE responses, RDMA READ response or ATOMIC Operation responses.

o9-19: If a TCA responder implements Reliable Connection service, or if a CA responder implements Reliable Datagram service, all acknowledge packets shall be strongly ordered. That is, all previous RDMA READ responses and ATOMIC responses shall be injected into the fabric before subsequent SEND, RDMA WRITE responses, RDMA READ response or ATOMIC Operation responses.

C9-28: A responder shall execute SEND requests, RDMA WRITE requests and ATOMIC Operation requests in the message order in which they are received. If the request is for an unsupported function or service, the appropriate response (for example, a NAK message, silent discard, or logging of the error) shall also be generated in the PSN order in which it was received.

- An application shall not depend upon the order of data writes to memory within a message. For example, if an application sets up data buffers that overlap, for separate data segments within a message, it is not guaranteed that the last sent data will always overwrite the earlier.

C9-29: The completion at the receiver is in the order sent (applies only to SENDs and RDMA WRITE with Immediate) and does not imply previous RDMA READs are complete unless fenced by the requester.

C9-30: A requester shall complete WQEs in the order in which they were transmitted.

C9-31: A work request with the fence attribute set shall block until all prior RDMA Read and Atomic WRs have completed.

C9-32: All WQEs shall be completed in the order they were posted independent of their execution order.

Due to the ordering rule guarantees of requests and responses for reliable services, the requester is allowed to write CQ completion events upon response receipt.

o9-20: An application shall not depend of the contents of an RDMA WRITE buffer at the responder until one of the following has occurred:

- Arrival and Completion of the last RDMA WRITE request packet when used with Immediate data.
- Arrival and completion of a subsequent SEND message.
- Update of a memory element by a subsequent ATOMIC operation.

o9-21: An application shall not depend on the contents of an RDMA READ target buffer at the requestor until the completion of the corresponding WQE.

o9-21.a1: An application shall not depend upon the contents of a SEND buffer at the responder until it has been completed.

C9-33: An application shall not depend on the contents of a receive queue buffer until the corresponding receive WQE has been completed.

9.6 PACKET TRANSPORT HEADER VALIDATION

Packet transport header validation is conducted on each packet that is passed up to the transport layer from the lower IBA layers. The purpose is to ascertain that the inbound packet can be associated with a particular queue pair. If it cannot, the packet is silently discarded. Packet transport header validation applies only to packets using the IBA transport.

C9-34: The transport layer shall validate the packet headers of all packets using the IBA transport according to the requirements in this section (9.6 Packet Transport Header Validation on page 269. A packet shall be deemed to be using the IBA transport if the msb of the LRH:LNH field is set to 1. If the msb of the LRH:LNH field is set to zero, then the packet is a raw packet. Raw packets are described in Section 9.8.4 Raw datagrams on page 394.

C9-35: For each inbound packet using the IBA transport, a CA shall validate the packet according to the state diagram shown in Figure 81[9]. The details of the state diagram are discussed in the remainder of this section.

If the packet can be associated with a given queue pair, further validation is conducted by comparing certain characteristics of the packet with context information stored with the queue pair (or EE Context, in the case of reliable datagram service). This level of validation is described in Section 9.7 Reliable Service on page 280 and Section 9.8 Unreliable Service on page 375.

Throughout this section, the phrase “packet is silently dropped” is used. The responder, unless otherwise noted, behaves as follows when a silent drop occurs:

- No acknowledge message is returned.
- No receive WQE is consumed by the responder.
- The errant request packet is not executed.
- Any request packets received prior to the errant request are executed and completed normally.
- Responder does not update its expected PSN.
- Responder resumes waiting for a valid inbound request packet.

The requester, unless otherwise noted, behaves as follows when a silent drop occurs:

- No send WQEs are completed as a result of a packet that is silently dropped.
- No direct action is taken as a result of the silently dropped packet, although error counters may be incremented or other similar events may occur.

9. The LVer field of the LRH is verified in the link layer before a packet is presented to the transport layer. The ICRC and VCRC headers are also verified in the link layer before a packet is presented to the transport layer. A packet with an invalid LVer field, invalid ICRC or invalid VCRC is dropped silently before reaching the transport layer.

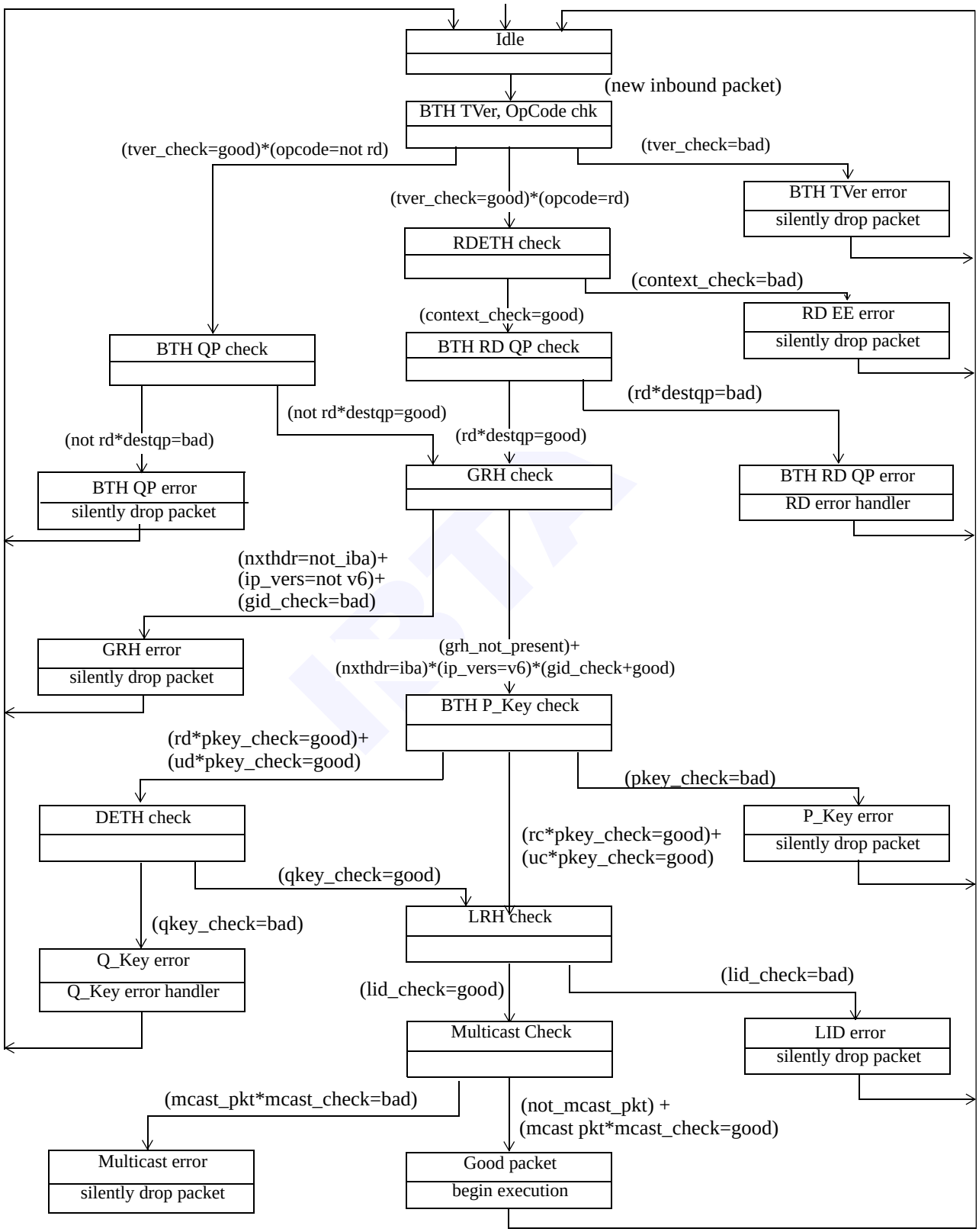


Figure 81 Packet Header Validation Process

- The silently dropped packet shall not count for purposes of satisfying the transport timer.

The queue state is not be changed. In addition, for connected transport services or reliable datagram, the connection or EE context is not torn down.

9.6.1 VALIDATING HEADER FIELDS

This section specifies the headers and fields that must be validated by a receiver of an inbound packet (either a request or response packet) before it can rely on the integrity of the packet.

9.6.1.1 BTH CHECKS

This section describes the fields of the BTH that must be validated for all incoming packets.

9.6.1.1.1 BTH:TVER VALIDATION

C9-36: The transport shall verify that the version number of the transport headers is supported by the CA or router. If the CA, switch or router does not support the indicated version, the packet shall be silently dropped. The only valid transport version is zero.

tver_check

- good: TVer field of BTH is 0x0
- bad: TVer field of BTH is non-zero

9.6.1.1.2 BTH:DESTINATION QP, OPCODE CHECK

Since the OpCode contained in the BTH of the inbound packet is used to determine if the selected destination QP is valid, OpCode validation is combined with validating the destination QP and its current condition.

C9-37: The transport shall verify that the destination QP exists and that the QP state is valid for receiving the inbound packet.

o9-22: This compliance statement is obsolete and replaced by o9-23.2.1:

o9-23: For CAs which support Unreliable Datagram Multicast, the destination QP value of 0xFFFFFF shall only be valid if there is at least one locally managed QP which is configured for IBA Unreliable Datagram Multicast service.

C9-38: BTH:OpCode[7:5] shall be checked to ensure that the service requested (RC, UC, RD, UD) is consistent with the configuration of the destination QP.

The response to an inbound packet which contains either an invalid destination QP, or whose destination QP is not in a valid state for receiving

the inbound packet, is dependent on the service being requested. This is determined by examining BTH:OpCode[7:5], which indicates whether the requested service is RC, RD, UC or UD.

Furthermore, if BTH:OpCode[7:5] indicates that the packet is for RD service, then the remainder of the OpCode bits must be examined to determine if the inbound packet is a request or a response packet.

C9-39: If BTH:OpCode[7:5] indicates that the packet is for RC, UC or UD services, and the destination QP does not exist, or the destination QP is not configured to provide the requested service, or the destination QP state is invalid, then the inbound packet is silently dropped.

C9-40: This compliance statement is obsolete and replaced by o9-23.2.1:

o9-23.2.1: For a CA that implements RD service: if BTH:OpCode[7:0] indicates an RD request packet, (SEND, RDMA READ Request, etc.), and the EE Context is valid, and the destination QP does not exist, or the destination QP is not configured to provide RD service, or the destination QP state is invalid, then a NAK-invalid RD request must be returned. If BTH:OpCode[7:0] indicates an RD response packet (RDMA READ Response, Acknowledge, etc.), and the destination QP does not exist, or the destination QP is not configured to provide RD service, or the destination QP state is invalid, then the inbound packet shall be silently dropped.

Table 37 Verification of Destination QP

Error Condition	Description
Invalid Destination QP identifier	No such QP exists on this CA. If the QP identifier is the IBA unreliable multicast QP (0xFFFFFF), there is no QP configured for IBA unreliable multicast service on this CA.
Incorrect Destination QP Configuration	The destination QP configuration is inconsistent with the service requested by BTH:OpCode[7:4].
Request packet: QP is not in Ready-to-Send state, Send-Queue-Drain state, or Ready-to-Receive state or Send-Queue-Error state.	Receive queue is not in a proper state to accept an inbound request packet.
Acknowledge packet: QP is not in Send-Queue-Drain state, Ready-to-Send state, or Send-Queue-Error state.	Send queue is not in a proper state to accept an inbound response packet.

destqp_check

- good: destination QP specified in BTH is a valid QP, and it is in the correct state to receive the packet, and the configuration of the QP is consistent with the service being requested.

- bad: destination QP specified in BTH does not exist, or is not in the correct state to receive the packet, or is configured inconsistently with the service being requested.

9.6.1.1.3 BTH:P_KEY

C9-41: If the destination QP is QP0, the BTH:P_Key shall not be checked.

C9-42: If the destination QP is QP1, the BTH:P_Key shall be compared to the set of P_Keys associated with the port on which the packet arrived. If the P_Key matches any of the keys associated with the port, it shall be considered valid.

C9-43: For all destination QPs other than QP0 or QP1, for all transport services except Reliable Datagram, the P_Key shall be compared with the P_Key associated with the responder's receive queue. An invalid P_Key shall cause the request packet to be silently dropped.

o9-24: For Reliable Datagram, the P_Key shall be compared with the P_Key associated with the responder's EE Context. An invalid P_Key shall cause the request packet to be silently dropped.

For further details of the process for matching the P_Key, please see 10.9.3 Partition Key Matching on page 526.

pkey_check

- good: BTH:P_Key matches value associated with recv queue or EE Context
- bad: BTH:P_Key does not match value associated with recv queue or EE Context

9.6.1.2 GRH CHECKS

This sections describes the fields of the GRH, if present, that must be validated.

Prior to receiving an inbound packet, a QP or EEC is configured as to whether or not a GRH is expected in each packet received on this connection. The mechanism by which this configuration occurs is outside the scope of the specification. One possible implementation is that this configuration occurs at connection establishment time via the Subnet Local bit in the CM REQ message. An inbound packet is only guaranteed to pass the GRH check if the presence or absence of the GRH is consistent with the QP (or EEC) configuration. For RC, RD and UC services, if an inbound packet arrives that is not consistent with the QP (or EEC) configuration with respect to the presence or absence of the GRH, the packet should be dropped. For UD services, since it is impractical to predict the source of a packet, the presence or absence of the GRH should not be checked.

As specified in Section, 9.6.1.5.2 IBA Unreliable Multicast Checks on page 280, a multicast packet must include a GRH.

C9-43.1.1: For UD services, if the packet is a multicast packet as specified in Section 9.6.1.5.2 IBA Unreliable Multicast Checks on page 280 and a GRH is not present in the packet, the packet shall be silently dropped.

C9-43.1.2: For RC, RD and UC services, if a received packet is consistent with the configuration of the QP (or EEC) with respect to the presence or absence of a GRH, then the packet shall be considered to have passed the GRH check, subject to the remaining GRH checks described in the rest of Section 9.6.1.2 GRH Checks on page 274.

9.6.1.2.1 GRH:NEXT HEADER

C9-44: If there is a GRH present, the Next Header field of the GRH must be checked. The value of the Next Header field should be set to 0x1B. Any other value indicates that this packet does not use the IBA transport, and the packet shall be silently dropped.

nxthdr_check

- good: GRH:NxtHdr field indicates IBA transport
- bad: GRH:NxtHdr field indicates non-IBA transport

9.6.1.2.2 GRH:IPVERS

C9-45: If there is a GRH present, the version field of the GRH shall be checked. If the version number is anything other than 6, the packet shall be silently dropped.

ip_vers

- not_v6: invalid GRH version number
- v6: GRH version number is valid

9.6.1.2.3 GRH:SGID, GRH:DGID

Connection Management is responsible for loading the primary SGID and DGID in the transport layer. If the given CA supports automatic path migration, a set of alternate SGID and DGID are also loaded. Primary and alternate GID comparison and actions are per the rules defined in Section 17.2.8 Automatic Path Migration on page 1031.

If a GRH is present, the SGID and DGID shall be verified as follows:

C9-46: If the destination QP is configured for UD transport service, the SGID shall not be validated at the transport layer. The DGID shall only be validated if the packet is a valid multicast packet. See 9.6.1.5.2 IBA Unreliable Multicast Checks on page 280 for a definition of a valid multicast packet.

C9-47: This compliance statement is obsolete and replaced by C9-47.1.1:

C9-47.1.1: If the destination QP is configured for RC, UC, or RD transport services, the SGID and the DGID shall be validated at the transport layer. Packets that do not pass these validity checks must be silently dropped.

The DGID is validated as follows:

1) If the DGID is set to the Reserved GID, the DGID is invalid.
2) If the DGID is set to the Loopback GID, the DGID is invalid.
3) If the DGID's scope indicates a Multicast GID but there is no locally associated QP, then the DGID is invalid.

Following these checks, the DGID is compared against the following. If it matches none of these, then the DGID is invalid.

4) The DGID is compared against the Primary DGID.
5) The upper 64-bits of the DGID is compared against the default GID prefix (0xFE80::0) and the lower 64-bits of the DGID is compared with the lower 64-bits of the Primary DGID
6) If Automatic path migration is supported, the DGID is compared with the Alternate DGID.
7) If Automatic path migration is supported, the upper 64-bits of the DGID is compared against the default GID prefix (0xFE80::0) and the lower 64-bits of the DGID is compared with the lower 64-bits of the Alternate DGID

The SGID is validated as follows:

1) If the SGID is set to multicast, the SGID is invalid.
2) If the SGID is set to the Loopback GID, the SGID is invalid.

Following these checks, the SGID is compared to the following. If the SGID does not match at least one of the following, it is invalid.

3) The SGID is compared against the Primary SGID
4) The upper 64-bits of the SGID is compared against the default GID prefix (0xFE80::0) and the lower 64-bits of the SGID is compared with the lower 64-bits of the Primary SGID
5) If Automatic path migration is supported, the SGID is compared with the Alternate SGID
6) If Automatic path migration is supported, the upper 64-bits of the SGID is compared against the default GID prefix (0xFE80::0) and the

lower 64-bits of the SGID is compared with the lower 64-bits of the Alternate SGID

gid_check

- good GRH SGID and DGID compared successfully
- bad GRH SGID or DGID is invalid

9.6.1.3 RDETH CHECKS

The section describes the fields of the RDETH, if present, that must be validated for reliable datagram service.

9.6.1.3.1 RDETH:EE CONTEXT

C9-47.2.1: If BTH:OpCode[7:5] indicates RD transport service and the CA does not implement RD service, then the packet must be silently dropped.

o9-25: If BTH:OpCode[7:5] indicates RD transport service, the RDETH shall be validated.

o9-26: The EE Context Identifier shall be verified per the rules in Table 38 Verification of EE Context on page 277. If the EE context is invalid, the packet must be silently dropped.

Table 38 Verification of EE Context

Error Condition	Description
Invalid Destination EE Context identifier	No such EE Context exists on this CA.
Request packet: EE Context is not in Ready-to-Send state, Send-Queue-Drain state or Ready-to-Receive state.	EE Context is not in a proper state to accept an inbound request packet.
Acknowledge packet: EE Context is not in Ready-to-Send state or Send-Queue-Drain state.	EE Context is not in a proper state to accept an inbound response packet.

context_check

- good: EE Context specified in RDETH is valid
- bad: EE Context specified in RDETH is invalid or CA does not support RD transport service

9.6.1.4 DETH CHECKS

This section describes the fields of the DETH, if present, that must be checked for datagram service, either reliable or unreliable.

9.6.1.4.1 DETH:Q_KEY

C9-48: If the destination QP is QP0, the DETH:Q_Key field shall not be validated.

C9-49: If the destination QP is QP1, the DETH:Q_Key field shall be considered valid if it compares successfully to the well-known Q_Key 0x80010000.

C9-50: For all packets received for a queue pair configured for datagram service, except QP0, the Q_Key shall be checked by the receiver's receive queue. If the Q_Keys do not match, the responder's behavior depends on whether the service is unreliable datagram or reliable datagram and shall be as follows:

Unreliable Datagram: the packet shall be silently dropped.

Reliable Datagram:

- A NAK-Invalid RD Request shall be returned.
- The P_Key used in the NAK may be supplied by the responder's EE Context or it may be extracted from the request packet being acknowledged.
- The PSN used in the NAK message is the PSN of the errant request packet.
- The EE Context's PSN is unchanged; it remains pointing to the failed request packet.
- The responder resumes waiting for a valid inbound request packet.

C9-51: The responder must not return an acknowledge message for a packet until the Q_Key and the P_Key for the packet have been checked by the receive queue.

qkey_check

- good: the Q_Key contained in the DETH matches that associated with the receive queue's stored Q_Key.
- bad: the Q_Key contained in the DETH does not match that associated with the receive queue's stored Q_Key.

9.6.1.5 LRH CHECKS

This section describes the fields of the LRH that must be validated for all inbound packets.

9.6.1.5.1 LRH:SLID, LRH:DLID

C9-52: The 16 bit fully resolved SLID and DLID contained in the LRH shall be validated.

C9-53: The DLID shall be validated only for reliable connected, unreliable connected or reliable datagram service. The DLID shall not be validated for Unreliable Datagram service.

C9-54: The SLID shall be validated only for reliable connected, unreliable connected or reliable datagram service. The SLID shall not be validated for Unreliable Datagram service.

To be valid, the SLID and the DLID contained in the LRH must compare exactly to one of the following:

1) Permissive LID
2) Multicast LID (for DLID only)
3) Primary LID
4) Alternate LID

C9-55: The permissive LID shall only be accepted as valid if the destination queue pair is QP0.

C9-56: If the SLID is a multicast LID, it shall be invalid.

C9-57: In an HCA configured for Reliable Connection or Unreliable Connection service, if an invalid LID is detected, the packet shall be silently dropped. For RC and UC service, this check is performed by the send or receive queue.

o9-27: If a TCA implements Reliable Connection or Unreliable Connection service and an invalid LID is detected, the packet shall be silently dropped. For RC and UC service, this check is performed by the send or receive queue.

o9-28: If a CA implements Reliable Datagram service, and if an invalid LID is detected, the packet shall be silently dropped. For RD service, this check is performed by the EE Context.

The primary SLID and DLID are stored in the QP or EE Context. If the given channel adapter supports transparent migration, an alternate SLID and DLID are also stored in the QP or EE Context as part of the alternate path. The choice of whether to compare the inbound SLID and DLID to the primary or alternate LIDs is a function of the current state of the automatic path migration state machine and the state of the MigReq bit in the BTH.

lid_check

- good: SLID and DLID contained in the LRH matches the SLID and DLID, respectively, stored in the QP or EE Context.
- bad: SLID or DLID contained in the LRH does not match the SLID and DLID, respectively, stored in the QP or EE Context.

9.6.1.5.2 IBA UNRELIABLE MULTICAST CHECKS

C9-58: A packet is declared to be an IBA unreliable multicast packet if the destination QP is 0xFFFFFF. To be considered valid, it must have the following three characteristics: The packet must contain a GRH, the DGID must be a valid multicast GID, and there must be at least one locally managed queue pair configured for multicast operation. If any of these conditions is not true, the packet is not a valid multicast packet and shall be dropped silently.

C9-59: The DGID shall be used to map the inbound packet to a particular locally managed QP.

multicast_check

- good: a multicast packet meets all the criteria cited above to be a valid multicast packet.
- bad: a multicast packet does not meet all the criteria cited above to be a valid multicast packet.

9.7 RELIABLE SERVICE

Reliable Service provides a guarantee that messages are delivered from a requester to a responder at most once, in order and without corruption. Key elements of the reliable service include a protection scheme to enable detection of corrupted data (CRC), an acknowledgment mechanism allowing the requester to ascertain that the message had been successfully delivered, a packet numbering mechanism to detect missing packets and to allow the requester to correlate responses with requests, and a timer to allow detection of dropped or missing acknowledgment messages.

This section addresses the acknowledgment and packet sequence numbering mechanisms. The CRC mechanism for detecting packet corruption is not addressed here. Note that CRCs are checked at lower protocol layers and may result in packets being dropped before they are delivered to the transport layer. These dropped packets may eventually be detected at the transport layer as sequence errors.

- Characteristics of reliable service
  - Messages delivered at most once, in order and without corruption in the absence of unrecoverable errors.
  - Each message is acknowledged either explicitly or implicitly.
- Types of reliable service
  - Reliable Connection
  - Reliable datagram

- Reliability mechanisms
  - ACK / NAK protocol
  - Packet Sequence Numbers (PSN)
- Responder considers operation complete when it has:
  - Received a valid “Last” or “Only” OpCode in the BTH,
  - Received all packets comprising the message in proper PSN order,
  - Payload has been committed to the local fault zone (SENDS or RDMA WRITEs),
  - Response has been committed to the wire for RDMA READs or ATOMIC Operations,
  - Acknowledge packet for the last packet of the request has been committed to the wire (including the appropriate fields for RDMA READ response)
- Requester considers the operation complete when:
  - All packets of the response (for RDMA READ or ATOMIC Operation) have been received and committed to local memory,
  - Acknowledge message has been received and validated.

C9-60: Before it can consider a WQE completed, the requester must wait for the necessary response(s) to arrive. If the requester requires an explicit response such that it can complete a given WQE, then the requester shall be responsible to take the necessary steps to ensure that the needed response is forthcoming.

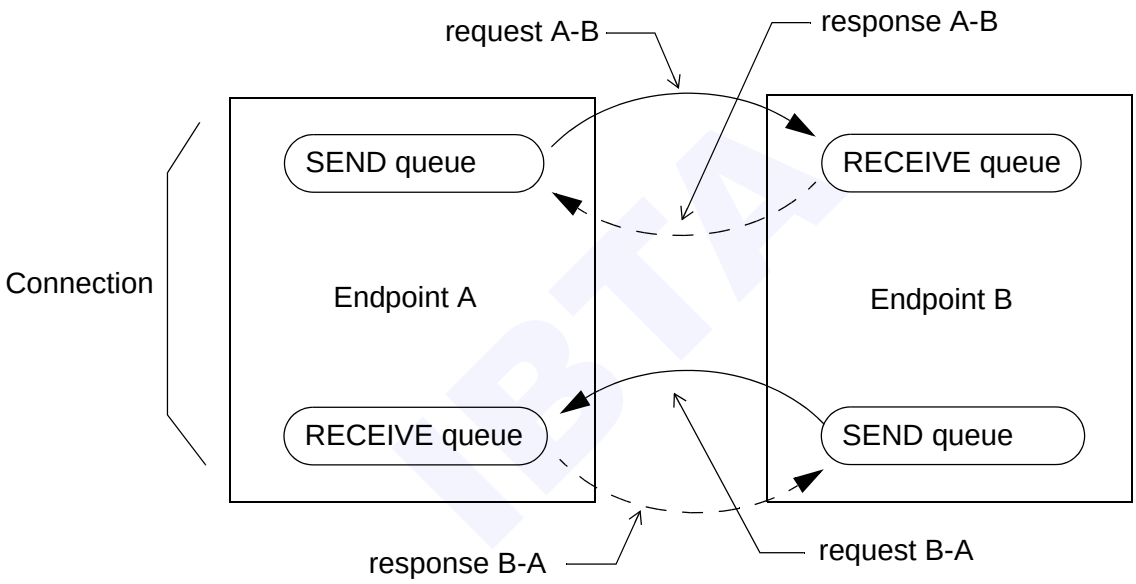
There are several mechanisms available to accomplish this, such as:

1) Set the AckReq bit on the last packet of every message, thus guaranteeing that the responder will generate the needed explicit response,
2) Set the AckReq bit on the last packet of the message for which an explicit response is desired,
3) If the AckReq bit was not set for the request for which an explicit response was desired, the requester can retry the request (with AckReq set) thus requiring the responder to return a response,
4) If the AckReq bit was not set for the request for which an explicit response was desired, the requester can send a NOP command (e.g. RDMA WRITE request with a length of zero) and set the AckReq bit. This strategy only works if the responder supports RDMA WRITES.

The choice of which of these, or other, strategies to use is implementation dependent.

9.7.1 PACKET SEQUENCE NUMBERS (PSN)

PSNs are transmitted within the Base Transport Header (BTH) for all packets. They are used to detect missing or out-of-order packets, and, for reliable services, to relate a response packet to a given request packet. Each IBA QP consists of a send queue and a receive queue; likewise, an EE Context has a send side and a receive side. There is a relationship between the PSN on a requester's send queue and the PSN on the responder's corresponding receive queue. Thus, each half of a QP (or EE Context) maintains an independent PSN; there is no relationship between the PSNs used on the Send queue and Receive queue of a given queue pair, or between different connection. This is illustrated in the figure below.



Request A-B and Response A-B (for reliable service) are related by PSN A-B. PSN A-B has no relationship to PSN B-A.

Figure 82 Send-Receive Queues Related by PSN

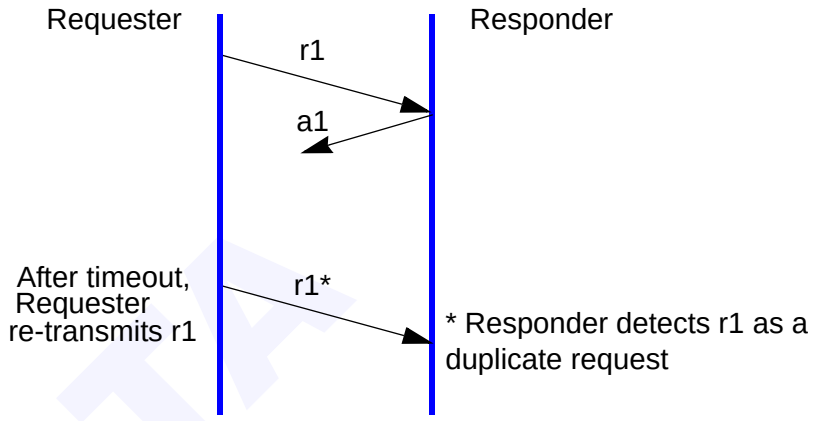
There are two abnormal conditions that must be detected and resolved at the responder to ensure reliable operation. The two conditions are duplicate packets and invalid packets.

1) Duplicate Packet. A duplicate packet may be recognized by the responder if the requester injects a request packet into the fabric more than once. This occurs when the requester detects a condition for which the prescribed recovery mechanism is to retry the operation.

   There are two primary causes of a timeout condition that may cause the requester to inject a given request packet into the fabric more than once:

- A response is late in arriving at the requester either because a response packet is lost or delayed in the fabric as shown in Figure 83 below, or because the responder experienced a delay in generating a response, or
- A request packet may be lost or delayed in reaching the responder as shown in Figure 85.

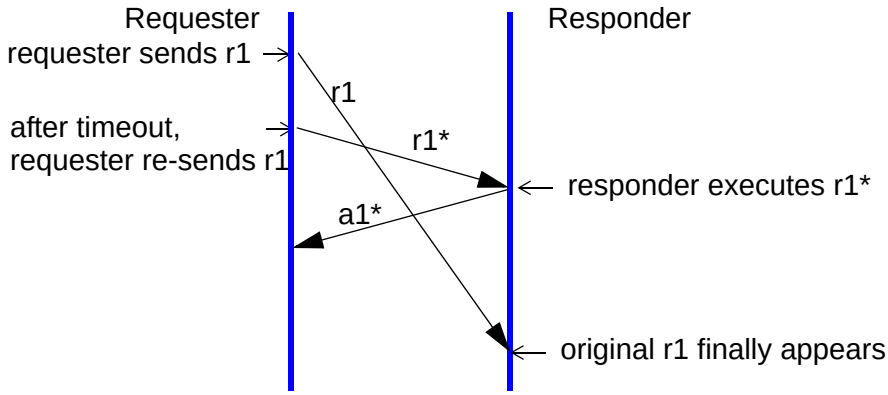
Regardless of the cause, the responder must be able to determine if an inbound request is a duplicate request that had been previously executed (or not) and respond appropriately.



'r' is a request packet,
'a' is an acknowledge packet

Figure 83 Duplicate Request Packets

In the previous figure, the response has been lost or delayed in the fabric causing the requester to detect a timeout condition and re-transmit the request. The responder interprets the second arrival of the request packet as a duplicate request.



'r' is a request packet,
'a' is an acknowledge packet

Figure 84 Ghost Request Packet

A duplicate packet may also be detected by the responder due to a "ghost" request packet. This occurs when a request packet is delayed in the fabric long enough to cause a timeout to occur at the requester. The requester re-sends the original request packet to which the responder generates the proper acknowledge message. Sometime later, the original (delayed) packet arrives at the responder which interprets the late arriving packet as a duplicate request. This may occur, for example, during automatic path migration.

2) Invalid Request Packet Sequence. This condition occurs when the responder believes that one or more request packets have been lost in the fabric. This is illustrated in the following figure.

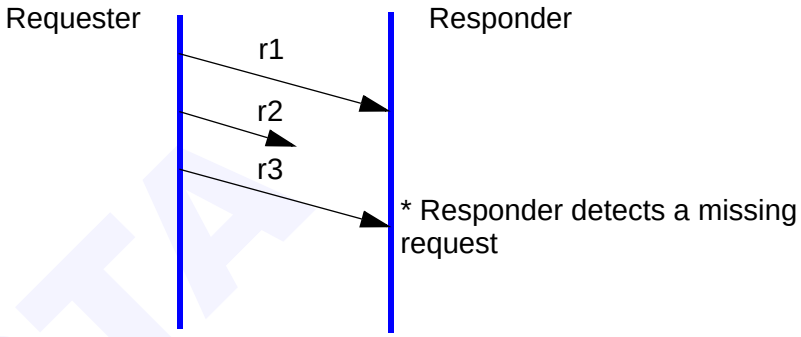


Figure 85 Lost Request Packet(s)

The distinction between an invalid packet and a duplicate packet is important since the requester's actions and the responder's actions are different for the two cases.

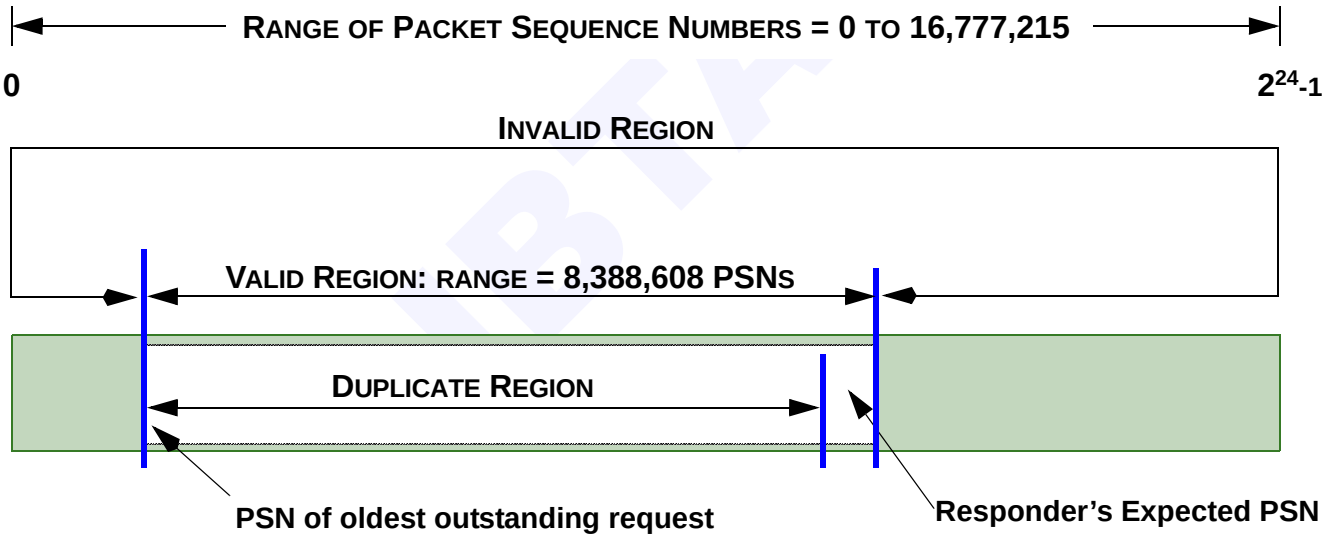
These two conditions must be detected both by the responder (for request packets), or by the requester (for response packets on reliable services).

In addition to duplicate packets and invalid packets, there is a third condition, called a Stale Packet ("TIME WAIT packet"). If a connection to a responder is torn down and a new connection is established while packets are in flight, a packet from the old (stale) connection may arrive at the responder. The responder, in turn, may interpret this stale incoming packet as a valid packet, when in fact it is a remnant of a previous connection. There are no transport layer mechanisms to guard against this condition; it is the responsibility of connection management to avoid re-using QPs until there is no possibility that a stale packet could arrive at the responder. This is done by placing the requester and responder QPs in a "Time Wait" state long enough to ensure that any stale packets left in the fabric have expired before re-using those QPs.

Duplicate packets are distinguished from invalid packets by the 24-bit PSN field which is carried in the base transport header, and allows room for uniquely naming up to 16,777,216 packets.

C9-61: In order to make it possible for the responder to distinguish duplicate packets from out of order packets, a given send queue shall have a series of PSNs no greater than 8,388,608 outstanding at any given time. Therefore, a send queue shall have no more than 8,388,608 packets outstanding at any given time. This includes the sum of all SEND request packets plus all RDMA WRITE request packets plus all ATOMIC Operation request packets plus all expected RDMA READ response packets.

Thus, the PSN space (consisting of a range of 16,777,216 PSNs) is divided into two regions, each occupying a range of 8,388,608 PSNs, called the valid region and the invalid region. This is illustrated in Figure 86.



A SEND QUEUE OR EE CONTEXT MAY HAVE NO MORE THAN 8,388,608 PACKETS OUTSTANDING AT ANY TIME.

Figure 86 Valid and Invalid PSN Regions

The responder further subdivides the valid region into an Expected PSN and a Duplicate region. The responder's expected PSN (ePSN) is defined in Section 9.7.4.1.2 Responder - PSN Verification on page 297, and is simply described as the PSN that the responder expects to find in the next new request packet to be received. The duplicate region is therefore defined to be the entire valid region, except for the single expected PSN. Simply put, a duplicate PSN is a PSN which the responder has seen and executed previously and which falls within the valid region.

9.7.1.1 PSN MODEL FOR RELIABLE SERVICE

C9-62: For an HCA requester using Reliable Connection service, the requester shall insert a PSN in each packet of each request it generates. When responding to the request, the responder shall insert a PSN in each packet of each response it generates.

o9-29: If a TCA implements Reliable Connection service, or if a CA requester implements Reliable Datagram service, the requester shall insert a PSN in each packet of each request it generates. When responding to the request, the responder shall insert a PSN in each packet of each response it generates.

Except for the special case of RDMA READ responses, there is a 1:1 relationship between the PSN in a request packet and the PSN in the corresponding response packet.

In the general PSN model, the requester calculates the PSN of the next request packet to be generated. This calculated PSN is called the Next PSN. At the time that the requester generates a new request packet, the “Next PSN” is copied into the BTH and thus becomes the current PSN. The requester then calculates a new “Next PSN”.

In order to detect missing or out of order packets, the responder also calculates the PSN it expects to find in the next inbound request packet. This is called the Expected PSN.

Conversely, when generating responses, the responder calculates the Response PSN to relate the response to a given request. However, due to acknowledge coalescing as described in 9.7.5.1.2 Coalesced Acknowledge Messages on page 308, the requester cannot necessarily predict which one of a range of PSNs may appear in the next response packet. Therefore, the requester must be prepared to accept any one of a range of Response PSNs. The range is bounded by the PSN of the oldest unacknowledged request packet and the expected response PSN of the most recently sent request packet. The requester evaluates the PSN of an inbound response packet to ensure that it falls between these two extremes. This general model is illustrated below.

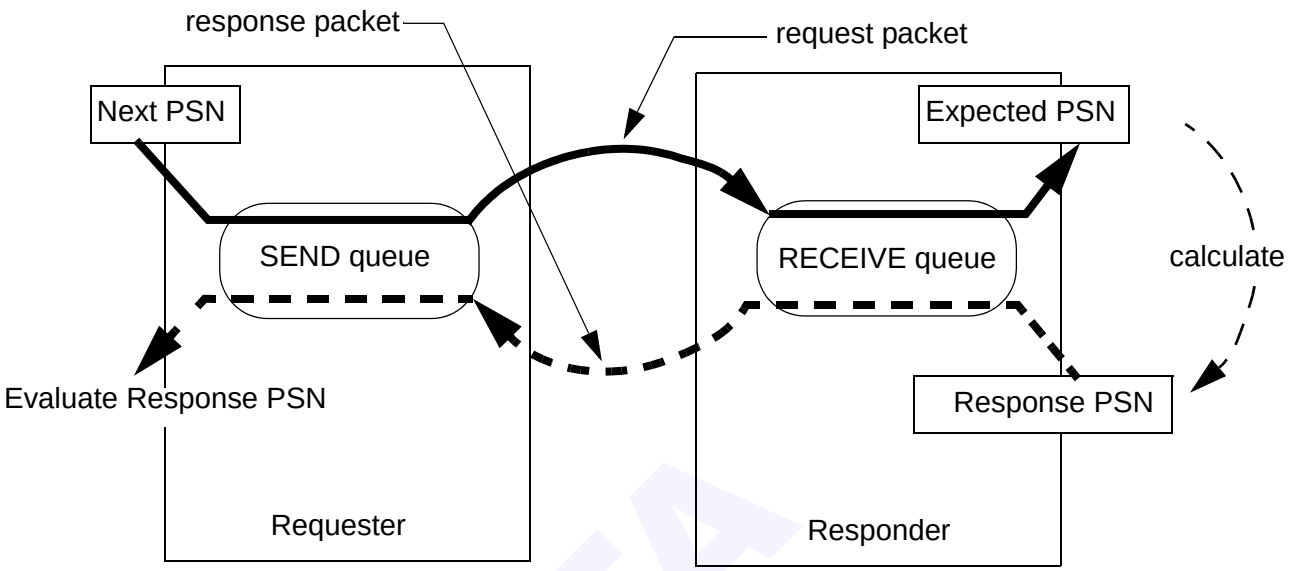


Figure 87 Request/Response PSNs

In the following sections only rarely is it not obvious from the context to which of the four PSNs the text is referring. Thus, it is common practice to refer to “PSN”, or “expected PSN” or some other variant. In the cases where the context is not clear, the above expressions are used for clarity.

9.7.2 ACK/NAK PROTOCOL

The ACK/NAK protocol, along with packet sequence numbers, is a fundamental component of reliable service, and applies to both reliable connected service and reliable datagram service. This and the following sections describe the protocol, provide a set of rules governing generation of ACK and NAK responses, specify the ACK and NAK codes and specify the requester’s required responses when it receives either an ACK or a NAK response.

The purpose of the ACK/NAK protocol is to allow the requester to ascertain deterministically if the responder correctly received the request packet. There are also mechanisms provided to ensure that a complete message was received correctly. This is accomplished through a combination of the packet sequence number and packet OpCodes (first/middle/last/only packet indications).

Since a response packet(s) can get lost in the fabric, the ACK/NAK protocol requires a requester to implement a timer to detect lost response packets. The transport timer is also described in this section.

The word “acknowledge” is used consistently throughout this section to mean either a negative (NAK) or a positive (ACK) acknowledgment. The generic term “response” is used to describe the acknowledgment returned by the responder to the requester. A response is carried in one or more acknowledge packets and may comprise, depending on the original request message, an ACK packet, a NAK packet, a RDMA READ response or an ATOMIC Operation response.

The following is a summary of the rules governing the ACK/NAK protocol:

- Each request packet received on a reliable service shall be acknowledged.
- Each RDMA READ request requires an explicit response. A RDMA READ response, with a properly formed ACK Extended Transport Header (AETH) is considered a valid response. The ACK Extended Transport Header appears in the first packet and last packet (or only packet) of a RDMA READ response. The details are covered below in Section 9.7.5.1.9 RDMA READ Responses on page 322
- Each ATOMIC Request requires an explicit response. An acknowledge packet, with a properly formed ACK Extended Transport Header (AETH) and an ATOMIC ACK Extended Transport Header (AtomicAckETH) is considered to be a valid response.
- Acknowledges may be coalesced; that is, a single acknowledge packet can serve as acknowledgment for one or more previous request packets.
- Acknowledge packets shall be returned in the PSN order in which the original request packet was received, including RDMA READ responses.
- A RDMA READ response consists of one or more packets; all other responses consist of exactly one packet.

C9-63: For an HCA responder using Reliable Connection service, the responder shall behave as follows. A responder shall acknowledge each request packet received. A responder shall generate an explicit response for each RDMA READ request received. A responder shall generate an explicit response for each ATOMIC Request received. A responder shall generate response packets in the PSN order in which the original request packets were received, including RDMA READ responses.

o9-30: If a TCA responder implements Reliable Connection service, or if a CA responder implements Reliable Datagram service, the responder shall behave as follows. A responder shall acknowledge each request

packet received. A responder shall generate an explicit response for each RDMA READ request received. A responder shall generate an explicit response for each ATOMIC Request received. A responder shall generate response packets in the PSN order in which the original request packets were received, including RDMA READ responses.

9.7.3 REQUESTER: GENERATING REQUEST PACKETS

This section specifies the requirements placed on a requester as it generates request packets.

9.7.3.1 REQUESTER SIDE - GENERATING PSN

C9-64: For Reliable Connection service in an HCA, the requester must place a value, called the current PSN, in the BTH:PSN field of every request packet.

o9-31: If Reliable Datagram service is implemented in a CA, or if Reliable Connection service is implemented in a TCA, then the requester must place a value, called the current PSN, in the BTH:PSN field of every request packet.

During connection establishment, the transport layer's client programs the next PSN to any value between zero and 16,777,215. For proper operation, the initial expected PSN value on the responder side must be loaded with the same value.

C9-65: For Reliable Connection service in an HCA, the initial PSN, as programmed by the transport layer's client, is the PSN that shall appear in the first request packet generated by the requester.

o9-32: If Reliable Datagram service is implemented in CA, or if Reliable Connection Service is implemented in a TCA, then the initial PSN, as programmed by the transport layer's client, is the PSN that shall appear in the first request packet generated by the requester.

Thereafter, the requester calculates the next PSN. The calculation depends on the operation being performed (SEND, RDMA READ, etc.) and the size of the data payload.

With one exception, the requester shall increment the current PSN value by one for each request packet it generates. The single exception is for any request packet immediately following a RDMA READ request message. In this case, the request packet immediately following the RDMA READ request message shall have a PSN that is one greater than the PSN of the last expected RDMA READ response packet. In this way, the requester leaves a "hole" in the PSN sequence of the request packets, such that all response packets will have monotonically increasing PSNs.

Thus, for RDMA READ Requests:

Let curr_PSN = PSN of a RDMA READ Request

Let next_PSN = PSN of the request following a RDMA READ Request

Let n = the number of expected RDMA READ response packets

Then next_PSN = (curr_PSN + n) modulo 2^{24}

Table 39 Requester's Calculation of Next PSN

Current Request Packet	PSN for Next Request Packet
SEND, RDMA WRITE, ATOMIC Operation	current PSN + 1 (modulo 2^{24})
RDMA READ	current PSN + (number of expected RDMA READ response packets) (modulo 2^{24})

Since the requester knows both the total length of the requested RDMA READ data and the PMTU between the requester and the responder, and since there is a requirement that each response packet (except a last or only packet) be filled to the full PMTU size, the requester can calculate the total number of expected response packets and thus calculate the PSN of the request immediately following the RDMA READ request.

C9-66: For an HCA requester using Reliable Connection service, the requester shall behave as follows. For each request packet other than the packet immediately following an RDMA READ request message, the requester shall increment the next PSN value by one modulo 2^{24}. For any request packet immediately following a RDMA READ request message, the packet shall have a PSN that is one greater (modulo 2^{24}) than the PSN of the last expected RDMA READ response packet.

o9-33: If a TCA requester implements Reliable Connection service, or if a CA requester implements Reliable Datagram service, the requester shall behave as follows. For each request packet other than the packet immediately following an RDMA READ request message, the requester shall increment the next PSN value by one modulo 2^{24}. For any request packet immediately following a RDMA READ request message, the packet shall have a PSN that is one greater (modulo 2^{24}) than the PSN of the last expected RDMA READ response packet.

9.7.3.2 REQUESTER - SPECIAL RULES FOR RELIABLE DATAGRAM

9.7.3.2.1 RDD CHECKING

For reliable datagram service, any given send queue is associated with an EE Context by a Reliable Datagram Domain (RDD). Each send queue and EE Context has a single RDD associated with it. Before sending a request, the EE context must check the RDD of the currently active send queue. If the send queue's RDD does not match the EE Context's RDD, the current message transfer is terminated and a timeout condition is indicated to the send queue.

o9-34: For each request, the requester must confirm that the RDD of the currently active send queue matches the RDD of the selected EE context.

o9-35: This compliance statement is obsolete and has been removed. Error behavior for RDD mismatch is defined in Table 56 Requester Side Error Behavior on page 401

9.7.3.2.2 RESYNC GENERATION

Under some conditions, a requester's EE Context is required to generate a special form of a request packet called a RESYNC request. This occurs when the requester EE Context elects to discontinue (abort) the current request message. The process of aborting the current request message and generating the subsequent RESYNC request is described in 9.7.8 Reliable Datagram on page 358 in the subsection dealing with handling QP errors.

The RESYNC request has the special property that it forces the responder to re-initialize its expected PSN to a value defined by the requester side EE Context.

The following paragraph governs the PSN that the requester is required to use when aborting the current message and generating the RESYNC request. It also governs the PSN it should expect in an acknowledgement received in response to the RESYNC request. These PSNs are identified in Figure 87 Request/Response PSNs on page 287 as the Next PSN and the Response PSN.

o9-35.a1: For a CA which supports Reliable Datagram service, when aborting the current request message and generating a RESYNC request, the requester side EE Context shall set the PSN of the RESYNC request to an increment of one greater (Modulo 2^{24}) than the largest PSN (Modulo 2^{24}) used in transmitting the current request. In addition, the requester shall reset the PSN it expects in a response to the same value. "The largest PSN used in transmitting the current request" means;

1) The (logically) largest PSN (Modulo 2^{24}) assigned to any packet of the request message including any retries of the request message, or
2) the PSN of the last expected RDMA READ response, if the request was an RDMA READ request.

For example, if the current request being transmitted consists of three packets with PSNs numbered 4, 5 and 6, then the PSN of the RESYNC request would be set to 7.

Or, consider the case where the requester is sending a large request message consisting of many packets. Even if the responder returns a NAK early in the transfer but the requester has sent subsequent packets, the PSN of the RESYNC request must be one greater (Modulo 2^{24}) than the PSN of any request packet sent, regardless of when the NAK packet was generated or its PSN.

In another example, if the current request being transmitted is an RDMA READ request with a PSN of 24, and to which the requester expects to receive 5 response packets, then the PSN of the RESYNC request would be set to 29, which is one greater than the PSN of the last expected RDMA READ response packet.

If a requester performs one or more retries, due to timeouts or other reasons, the PSN of the RESYNC request must be one greater (Modulo 2^{24}) than the PSN of any request packet sent, or response packet expected, for all of the previous attempts.

o9-35.a2: For a CA which supports Reliable Datagram service, the requester shall insert in the RESYNC request source and destination QPns which are identical to the source and destination QPs from the current request (which is being aborted).

9.7.3.3 REQUESTER - GENERATING OPCODES

The opcodes generated by a requester must fit into a schedule of opcodes as shown below.

C9-67: A requester must generate packet opcodes which fit within the schedule of valid OpCode sequences as shown in Table 40 Schedule of Valid OpCode Sequences on page 293.

Table 40 Schedule of Valid OpCode Sequences

Previous Packet OpCode	Valid OpCodes for Current Packet
None e.g., first packet following connection establishment	"First" packet "Only" packet
"First" packet	"Middle" packet (message is 3 or more packets) "Last" packet (message is exactly 2 packets) Type of operation must match the previous OpCode
"Middle" packet	"Middle" packet "Last" packet Type of operation must match the previous OpCode
"Last" packet	"First" packet (1st packet of a new message) "Only" packet (1st packet of a new single packet msg)
"Only" packet	"First" packet "Only" packet

C9-68: When generating a request packet, the BTH:Opcode shall be as specified in Table 35 OpCode field on page 235.

9.7.3.4 REQUESTER - GENERATING PAYLOADS

The requester shall generate payload lengths as a function of the opcode as follows:

C9-69: If the OpCode specifies a "first" or "middle" packet, then the packet payload length must be a full PMTU size.

C9-70: If the OpCode specifies a "only" packet, then the packet payload length must be between zero and PMTU bytes in size. Thus, the only way to create a zero byte length transfer is by use of a single packet message.

C9-71: If the OpCode specifies a "last" packet, then the packet payload length must be between one and PMTU bytes in size.

C9-72: For an HCA, if the OpCode specifies an RDMA WRITE request, the length specified in the DMALen field of the RETH shall be no less than zero, and no greater than 2^{31} bytes.

o9-36: If RDMA WRITE is implemented in a TCA and the OpCode specifies an RDMA WRITE request, the length specified in the DMALen field of the RETH shall be no less than zero, and no greater than 2^{31} bytes.

9.7.4 RESPONDER: RECEIVING INBOUND REQUEST PACKETS

This section describes the process used by a responder when it receives an inbound request packet.

9.7.4.1 RESPONDER - INBOUND PACKET VALIDATION

C9-73: For Reliable Connection service in an HCA, inbound request packets shall be validated as shown in Figure 88 on page 295.

o9-36.a1: If Reliable Connection service is implemented in a TCA, inbound request packets shall be validated as shown in Figure 88 on page 295.

o9-37: If Reliable Datagram service is implemented in a CA, inbound request packets shall be validated as shown in Figure 89 on page 296.

The following sections describe each of the validation checks and the responder's behavior / response.

9.7.4.1.1 RESPONDER - SPECIAL RULES FOR RELIABLE DATAGRAM CHECKING

o9-38: For RD within a HCA, when an inbound packet arrives, the receive queue must test its own RDD value against that of the EE Context over which the inbound packet arrived. If they do not match, the receive queue must drop the packet and schedule a NAK-Invalid RD Request. The P_Key and PSN to be used for returning the NAK shall be supplied by the EE Context.

o9-38.a1: If Reliable Datagram service is implemented in a CA, a responder shall do the following when an inbound packet arrives:

1) If the responder receives a new request packet (i.e. PSN = expected PSN) with an opcode of "middle" or "last" (i.e., the responder has previously received a new request packet with an opcode of "first" and has not yet received a new request packet with an opcode of "last"), the responder shall validate that the source and destination QPns contained in the new request packet exactly match those of the most recently received "first" packet. If the source and destination QPns of the new "middle" or "last" request packet do not match those of the most recently received "first" request, the responder shall ignore the packet.
2) If the responder receives a duplicate request packet (i.e., the PSN is in the duplicate region), the responder shall validate that the source and destination QPns in the duplicate request packet exactly match those of the most recently received new request. If the source and destination QPns do not match those of the most recently received new request, the responder shall ignore the packet.

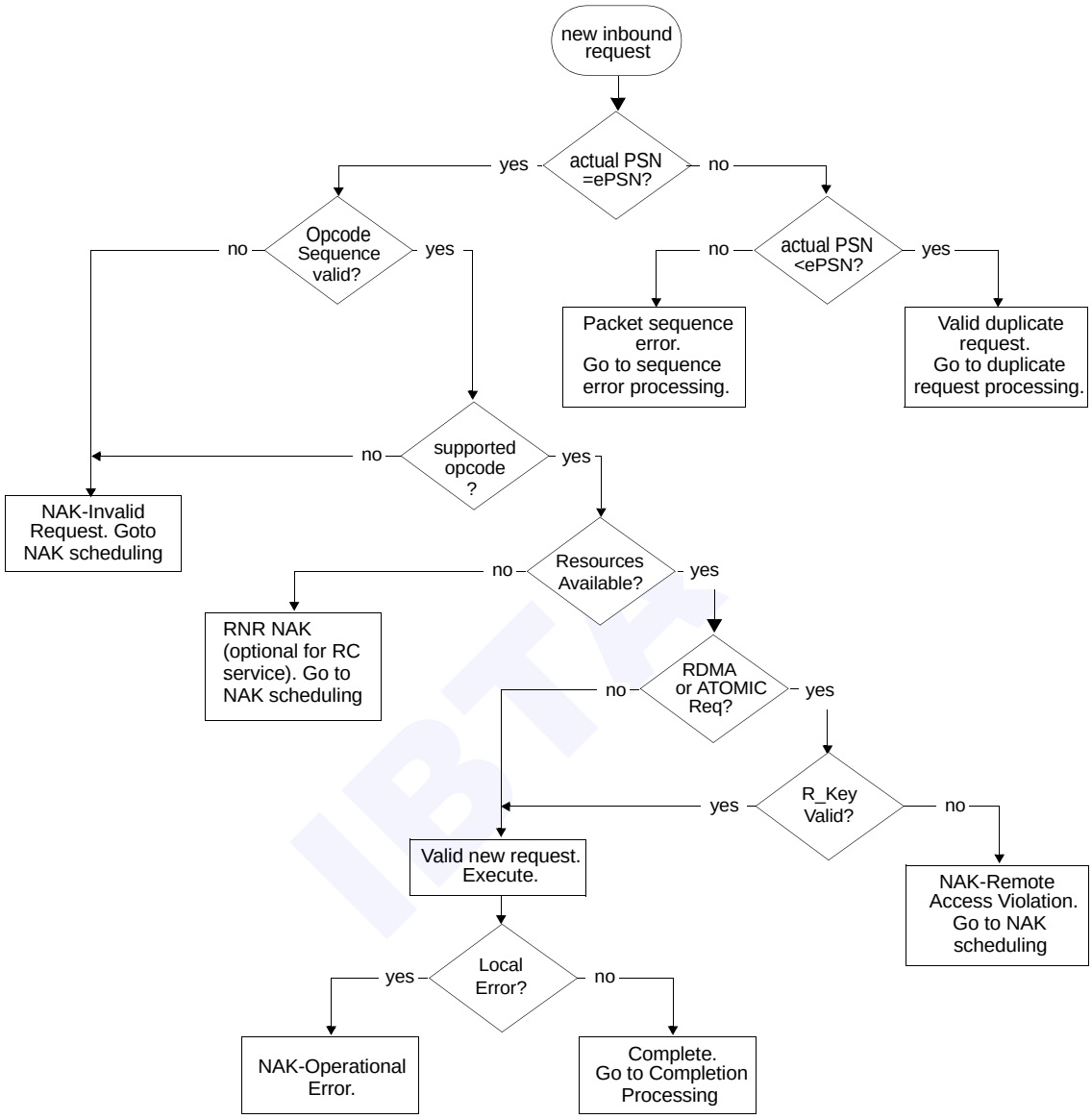


Figure 88 Inbound Request Packet Validation, RC mode

OM10528

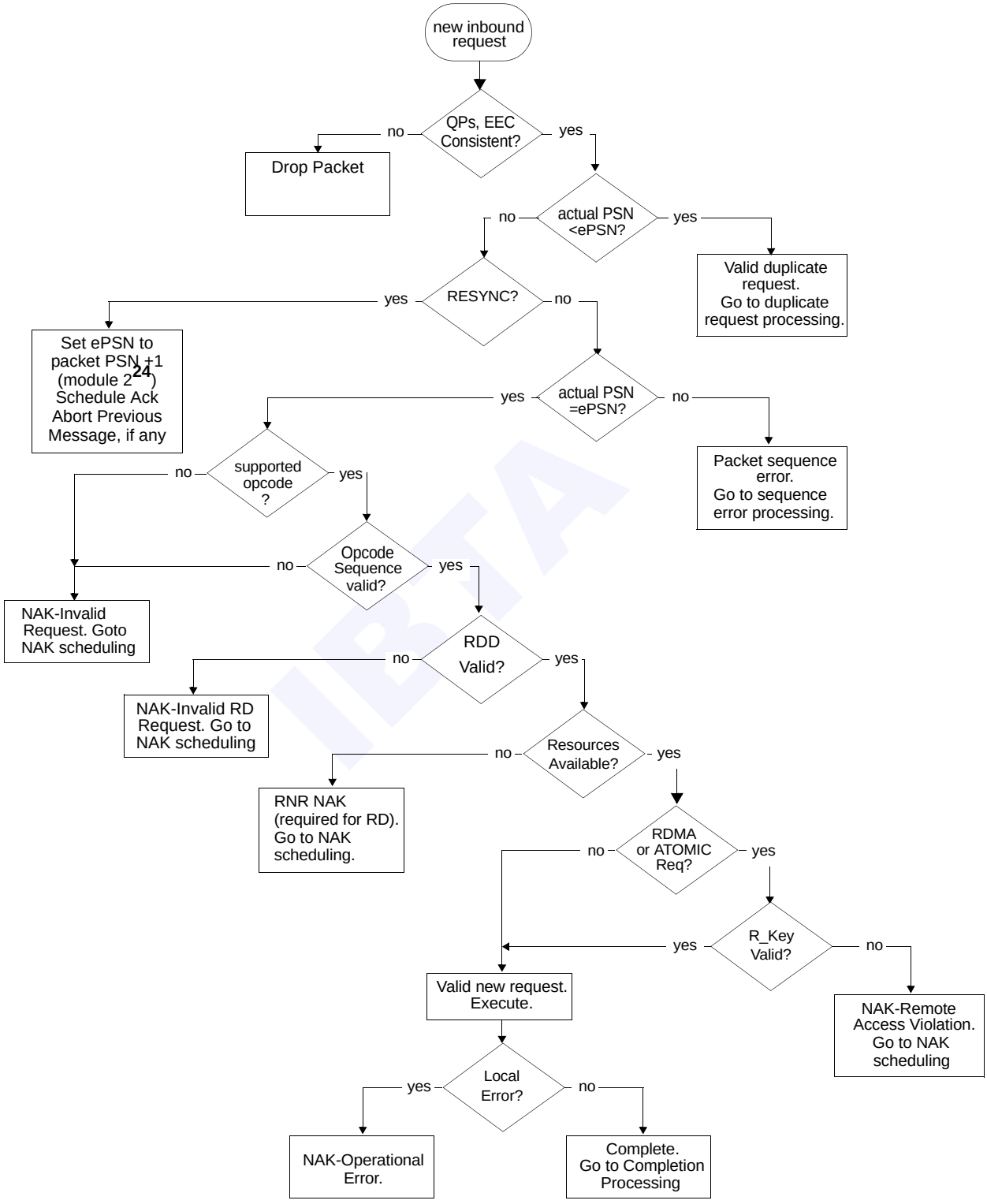


Figure 89 Inbound Request Packet Validation, RD mode

o9-38.a2: If Reliable Datagram service is implemented in a CA, when an inbound RESYNC request arrives, the responder shall do the following:

1) If the responder receives a RESYNC request, it shall accept the request regardless of the value of the source and destination QPns in the RESYNC request.
2) If the responder receives a RESYNC request, it shall accept that request regardless of the state of the opcode sequence (“first”, “middle”, etc.) of the currently executing request (i.e. the most recently received new request).
3) If the PSN of the RESYNC request is equal to or logically greater than the responder’s expected PSN (i.e. the RESYNC request is a new request), the responder shall:
   a) Set its expected PSN equal to the PSN of the RESYNC request plus one (Modulo 2^{24}),
   b) Use the PSN of the RESYNC request as the PSN of the response,
   c) Abort any processing associated with the currently executing request if the currently executing request is incomplete (i.e. a Request packet with an opcode of “last” or “only” has not yet arrived). If such a currently executing request was a SEND RDMA WRITE with Immediate (and thus would have consumed a receive WQE), the partially consumed receive WQE shall be completed in error.
4) If the PSN of the RESYNC request is logically less than the responder’s expected PSN, the responder shall treat the RESYNC request as a duplicate request and thus shall not change its expected PSN. The responder shall use the PSN of the duplicate RESYNC request as the PSN of the response.

9.7.4.1.2 RESPONDER - PSN VERIFICATION

C9-74: For Reliable Connection service in an HCA responder, and before executing the inbound request, the responder shall check the PSN by comparing the inbound BTH:PSN to the responder’s expected PSN. The PSN shall be checked by the responder’s receive queue.

o9-39: If Reliable Datagram service is implemented in a CA, or if Reliable Connection service is implemented in a TCA, and before executing the inbound request, the responder shall check the PSN by comparing the inbound BTH:PSN to the responder’s expected PSN. The PSN shall be checked by the responder’s receive queue.

For reliable datagram service, the PSN is checked by the responder’s EE Context.

To a large extent, the responder's behavior in responding to a request is based on an interpretation of the incoming PSN.

Logically, a receive queue or EE Context maintains an expected PSN (ePSN). This is the PSN that the responder expects to find in the BTH of the next new request packet it receives. The rules that the responder uses to calculate its next expected PSN are the same as those used by the requester when it calculates the PSN value to insert in its next request packet.

C9-75: For Reliable Connection service in an HCA responder, a responder shall use the rules given in 9.7.3.1 Requester Side - Generating PSN on page 289 to calculate its expected PSN.

o9-40: If Reliable Datagram service is implemented in a CA, or if Reliable Connection service is implemented in a TCA, a responder shall use the rules given in 9.7.3.1 Requester Side - Generating PSN on page 289 to calculate its expected PSN.

The responder's expected PSN is initialized at connection establishment time by the Communication Manager to any value between zero and 16,777,215. For proper operation, this initial value must match the initial next PSN value as loaded on the requester.

The initial expected PSN can only be set by the client when the queue is in the Initialized state. Attempts by the client to set the PSN when it is in any other state may be ignored by the transport layer.

C9-76: For Reliable Connection service in an HCA responder, the HCA shall update its expected PSN only when the receive queue is in a state such that it is properly conditioned to receive request packets. For example, the transport does not modify the expected PSN when the queue pair is in the Initialized state.

o9-41: If Reliable Connection service is implemented in a TCA, the responder shall update its expected PSN only when the Receive Queue is in a state such that it is properly conditioned to receive request packets. For example, the transport does not modify the expected PSN when the queue pair is in the Initialized state.

o9-42: If Reliable Datagram service is implemented in a CA, the responder shall update its expected PSN only when the EE Context is in a state such that it is properly conditioned to receive request packets.

When compared to its expected PSN, the actual PSN of an inbound request message may fall into one of three regions; it may be exactly equal to the responder's expected PSN, it may be logically "less" than the responder's expected PSN and thus fall into the duplicate region as shown

in Figure 86, or it may fall outside both the valid region and the expected PSN "region", and thus be invalid.

Expected (new) Request: An inbound request packet received with a PSN that exactly matches the responder's expected PSN is a new request packet.

C9-77: For Reliable Connection service in an HCA responder, a new request packet shall be validated normally and executed according to the rules governing order of execution. Once the request has been executed, a response shall be scheduled as specified in 9.7.5 Responder: Generating Responses on page 306.

o9-43: If Reliable Datagram service is implemented in a CA, or if Reliable Connection service is implemented in a TCA, a new request packet shall be validated normally and executed according to the rules governing order of execution. Once the request has been executed, a response shall be scheduled as specified in 9.7.5 Responder: Generating Responses on page 306.

Note that it is not required to return a discrete acknowledge packet for each inbound request packet.

Once a packet with a valid expected PSN has been received, the responder advances its expected PSN by calculating the new expected PSN, and slides the valid region window up to reflect the new range of valid PSNs.

Valid Duplicate Request: A PSN that falls within the valid region, but is not the expected PSN, is a valid duplicate request packet.

C9-78: For Reliable Connection service in an HCA responder, the responder shall respond to valid duplicate requests as specified in 9.7.5.1.4 Acknowledging Duplicate Requests on page 312.

o9-44: If Reliable Datagram service is implemented in a CA, or if Reliable Connection service is implemented in a TCA, then the responder shall respond to valid duplicate requests as specified in 9.7.5.1.4 Acknowledging Duplicate Requests on page 312.

Table 41 summarizes those actions.

Table 41 Summary: Responder Actions for Duplicate PSNs

Duplicate Request Message	Responder Action
SEND or RDMA WRITE or RESYNC	Schedule acknowledge packet
RDMA READ	Re-execute request, schedule response
ATOMIC Operation	Do not re-execute request, after validating the request, return the saved results from the referenced ATOMIC Operation request.

Invalid Request: A packet with an actual received PSN outside the valid region and not in the expected “regions” is an invalid request. An invalid PSN value is generally an indication that one or more request packets have been lost in the fabric.

The responder’s detailed behavior in response to an invalid request request packet is as follows:

- The errant request packet is not executed.
- Any request packets received prior to the errant request must be executed and completed before the NAK-Sequence Error is issued since it acts as an implicit ACK for prior outstanding SEND or RDMA WRITE requests, and as an implicit NAK for outstanding RDMA READ or ATOMIC Operation requests.
- Return a NAK-Sequence error to the requester.
- The responder does not update its expected PSN.

C9-79: For Reliable Connection service in an HCA responder, when the actual PSN of an inbound request message is outside the valid region (Invalid Request), a NAK-Sequence Error shall be returned by the responder. Any request packets received prior to the errant request must be executed and completed before the NAK-Sequence Error is issued.

o9-45: If Reliable Datagram service is implemented in a CA, or if Reliable Connection service is implemented in a TCA, and if the actual PSN of an inbound request message is outside the valid region (Invalid Request), then a NAK-Sequence Error shall be returned by the responder. Any request packets received prior to the errant request must be executed and completed before the NAK-Sequence Error is issued.

The responder resumes waiting for a valid inbound request packet that has a PSN equal to its expected PSN or within its valid region. If, while waiting for a valid new request, the responder receives any subsequent

invalid request packets, those packets are simply dropped silently; no NAK is returned.

C9-80: For Reliable Connection service in an HCA responder, after generating a NAK-Sequence Error, the responder shall not generate an ACK or NAK until it receives either a valid new request, or a valid duplicate request.

o9-46: If Reliable Datagram service is implemented in a CA, or if Reliable Connection service is implemented in a TCA, then after generating a NAK-Sequence Error, the responder shall not generate an ACK or NAK until it receives either a valid new request, or a valid duplicate request.

There is no requirement that the queue be stopped or for a connected transport service that the connection be torn down.

9.7.4.1.3 RESPONDER - OPCODE SEQUENCE CHECK

A request packet must fit within a schedule of valid OpCode sequences. For Reliable Connected and Reliable Datagram services the responder shall check the sequence of packet OpCodes comprising the request message as follows:

1) If this is the first packet following establishment of the connection, then the packet OpCode must indicate either “first” or “only”.
2) If the last valid packet received had an OpCode indicating “first”, then the current OpCode must indicate either “middle” or “last”. It must also match the operation type specified in the last valid packet (Send, RDMA, ATOMIC Operation). It is an error if the current OpCode indicates “first” or “only”, since that implies that the last packet of the previous message was missed.
3) If the last valid packet received had an OpCode indicating “middle”, then the current OpCode must indicate either “middle” or “last”. It must also match the operation type specified in the last valid packet (Send, RDMA, ATOMIC Operation). It is an error if the current OpCode indicates “first” or “only” packet since that implies that the last packet of the previous message was missed.
4) If the last valid packet received had an OpCode indicating “last”, then the current OpCode must indicate either “first” or “only”. It is an error if the current OpCode indicates either “middle” or “last”, since that implies that the first packet of the message was missed.
5) If the last valid packet received had an OpCode indicating “only”, then the current OpCode must indicate either “first” or “only”. It is an error if the current OpCode indicates either a middle packet or last packet since that implies that the first packet of the message was missed.

These rules are stated succinctly in the following table.

Table 42 Schedule of Valid OpCode Sequences

Previous Packet OpCode	Valid OpCodes for Current Packet
None e.g., first packet following connection establishment	"First" packet "Only" packet
"First" packet	"Middle" packet (message is 3 or more packets) "Last" packet (message is exactly 2 packets) Type of operation must match the previous OpCode
"Middle" packet	"Middle" packet "Last" packet Type of operation must match the previous OpCode
"Last" packet	"First" packet (1st packet of a new message) "Only" packet (1st packet of a new single packet msg)
"Only" packet	"First" packet "Only" packet

C9-81: For an HCA responder using Reliable Connected service, the responder shall check that the sequence of packet OpCodes comprising the request message conforms to the schedule shown in Table 42 Schedule of Valid OpCode Sequences on page 302. If the responder detects an invalid opcode sequence, it shall return a NAK-Invalid Request to the requester.

o9-47: If a TCA responder implements Reliable Connected service, the responder shall check that the sequence of packet OpCodes comprising the request message conforms to the schedule shown in Table 42 Schedule of Valid OpCode Sequences on page 302. If the responder detects an invalid opcode sequence, it shall return a NAK-Invalid Request to the requester.

o9-48: If a CA responder implements Reliable Datagram service, the responder shall check that the sequence of packet OpCodes comprising the request message conforms to the schedule shown in Table 42 Schedule of Valid OpCode Sequences on page 302. If the responder detects an invalid opcode sequence, it shall return a NAK-Invalid Request to the requester.

The detailed behavior in the presence of an invalid OpCode sequence is specified in Section 9.9 Error detection and handling on page 396.

o9-49: This compliance statement is obsolete and has been removed.

9.7.4.1.4 RESPONDER OPCODE VALIDATION

C9-82: Before executing an inbound request, the responder shall validate the OpCode field of the BTH.

The OpCode is checked for the following characteristics:

- The requested function (Send, RDMA, ATOMIC) is supported by this receive queue,
- If the request is for an RDMA READ or an ATOMIC Operation, there are sufficient resources available to receive it.

As the packet was passed up to the transport layer, BTH OpCode field[7:5] was checked to ensure that the requested operation was for a reliable service. If it was not, then the packet was silently dropped. This check is specified in Section 9.6 Packet Transport Header Validation on page 269. Thus, before the packet arrives at the queue pair for validation according to the rules in this section, it is already known to be a request for a reliable service.

C9-83: For Reliable Connection service in an HCA responder, if the request is for a function which this receive queue does not support, then a NAK-Invalid Request shall be returned.

For example, if the queue pair is not configured to accept requests for RDMAs, but the request is for an RDMA WRITE, then a NAK-Invalid Request shall be returned.

o9-50: If Reliable Datagram service is implemented in a CA, or if Reliable Connection service is implemented in a TCA, if the request is for a function which this receive queue does not support, then a NAK-Invalid Request shall be returned.

C9-84: For Reliable Connection service in an HCA responder, and the BTH OpCode field[4:0] specifies a Reliable Connection reserved opcode or a Reliable Datagram reserved opcode, a NAK-Invalid Request shall be returned.

o9-51: If Reliable Datagram service is implemented in a CA, or if Reliable Connection service is implemented in a TCA, and the BTH OpCode field[4:0] specifies a Reliable Connection reserved opcode or a Reliable Datagram reserved opcode, then a NAK-Invalid Request shall be returned.

C9-85: For Reliable Connection service in an HCA responder, if BTH OpCode field[4:0] specifies a first or middle request packet (e.g. SEND First, or RDMA WRITE Middle), then the pad count bits shall be verified to be b00, indicating no pad bytes are present. If the pad count bits are non-zero, a NAK-Invalid Request shall be returned.

o9-52: If Reliable Datagram service is implemented in a CA, or if Reliable Connection service is implemented in a TCA, if BTH OpCode field[4:0] specifies a first or middle request packet (e.g. SEND First, or RDMA

WRITE Middle), then the pad count bits shall be verified to be b00, indicating no pad bytes are present. If the pad count bits are non-zero, a NAK-Invalid Request shall be returned.

C9-86: For Reliable Connection service in an HCA responder, if there are insufficient resources to receive a new RDMA READ or ATOMIC Operation request, then a NAK-Invalid Request shall be returned.

o9-53: If Reliable Datagram service is implemented in a CA, or if Reliable Connection service is implemented in a TCA, and if there are insufficient resources to receive a new RDMA READ or ATOMIC Operation request, then a NAK-Invalid Request shall be returned.

The behavior for returning a NAK-Invalid Request is as follows:

- The errant request packet is not executed.
- Any request packets received prior to the errant request must be executed and completed before the NAK-Invalid Request is issued. This is important since the NAK effectively coalesces responses to earlier outstanding request and acts as an implicit response for prior outstanding SENDs, RDMA WRITEs, ATOMIC Operations or RDMA READ requests. See Section 9.7.5.1.2 Coalesced Acknowledge Messages on page 308 for details.
- NAK-Invalid Request is returned.
- The responder does not update its expected PSN.

C9-87: For Reliable Connection service in an HCA responder, any request packets received prior to a packet containing an invalid opcode must be executed and completed before a NAK-Invalid Request is issued by the responder.

o9-54: If Reliable Datagram service is implemented in a CA, or if Reliable Connection service is implemented in a TCA, then any request packets received prior to a packet containing an invalid opcode must be executed and completed before a NAK-Invalid Request is issued by the responder.

More detail on error behavior in the presence of an invalid request is given in Section 9.9.3 Responder Side Behavior on page 408.

9.7.4.1.5 RESPONDER R_KEY VALIDATION

A R_Key violation is caused by any or all of the following conditions for either a RDMA READ, RDMA WRITE, or ATOMIC Operation:

- The R_Key field of the RETH is invalid (for RDMA READ or WRITEs)
- The R_Key field of the AtomicETH is invalid (for ATOMIC Operations).

- The virtual address and length, or type of access specified, is outside the locally defined limits associated with the R_Key. For an RDMA WRITE request, the length check is conducted on a per packet basis, and is based on the LRH:PktLen field. For an RDMA READ request, the length check is based on the RETH:DMA Length field.
- For an HCA, an R_Key violation also includes a violation of the protection domain as defined in 10.2.3 Protection Domains on page 434.

C9-88: For an HCA responder using Reliable Connection service, for each zero-length RDMA READ or WRITE request, the R_Key shall not be validated, even if the request includes Immediate data.

o9-55: If an HCA responder implements Reliable Datagram service, or if a TCA responder implements Reliable Connection and RDMA functionality, or if a TCA responder implements Reliable Datagram service and RDMA functionality, the responder shall behave as follows. For each zero-length RDMA READ or WRITE request, the R_Key shall not be validated, even if the request includes Immediate data.

C9-89: If the responder's receive queue detects a R_Key violation, a NAK-Remote Access Error shall be returned to the requester using the PSN of the errant request packet.

C9-90: Any request packets received prior to a packet containing an R_Key violation shall be executed and completed before a NAK-Remote Access Error is issued by the responder.

See 9.7.5.2.4 Remote Access Error on page 326 for additional details.

9.7.4.1.6 RESPONDER - LENGTH VALIDATION[10]

C9-91: The PktLen field of the LRH shall be checked to confirm that there is sufficient space available in the receive buffer specified by the receive WQE. If the buffer defined by the receive WQE is insufficient to hold an inbound SEND request then a NAK-Invalid Request shall be returned.

C9-92: The length of the packet shall also be validated by comparing it to the OpCode as follows:

If the OpCode specifies a "first" or "middle" packet, then the packet payload length must be a full PMTU size.

If the OpCode specifies a "only" packet, then the packet payload length must be between zero and PMTU bytes in size. Thus, the only way to create a zero byte length transfer is by use of a single packet message.

10. CAs are not required to validate the GRH packet length.

If the OpCode specifies a “last” packet, then the packet payload length must be between one and PMTU bytes in size.

C9-93: If a packet is detected with an invalid length the request shall be an invalid request.

The responder’s behavior in such a case is specified in Section 9.9.3 Responder Side Behavior on page 408.

In addition to checking the LRH:PktLen field, the DMA Length field of the RETH is checked as follows.

For an RDMA WRITE request, the responder may optionally check the DMA Length field in the RETH to ensure that it does not specify a transfer length of greater than 2^{31} bytes. It may also, at the end of the transfer, verify that the sum of the packet payloads equalled the DMALen field in the RETH. If the responder detects either of these conditions, it may treat the request as an invalid request.

C9-94: For an HCA validating an inbound RDMA READ request, the DMA Length field shall be checked. If the request is for greater than 2^{31} bytes, then a NAK-Invalid Request shall be returned.

o9-56: If a TCA implements RDMA operations, then for an inbound RDMA READ request, the DMA Length field shall be checked. If the request is for greater than 2^{31} bytes, then a NAK-Invalid Request shall be returned.

9.7.4.1.7 RESPONDER LOCAL OPERATION VALIDATION

A valid inbound request may still fail to complete due to a failure that is local to the responder, e.g. local memory translation error while accessing local memory. All local responder errors are reported to the requester as NAK-Remote Operational Error. See 9.7.5.2.6 Remote Operational Error on page 327 for additional details.

9.7.5 RESPONDER: GENERATING RESPONSES

9.7.5.1 RESPONDER SIDE BEHAVIOR

This section specifies the required behavior that a responder must follow when generating acknowledge messages.

9.7.5.1.1 GENERATING PSNS FOR ACKNOWLEDGE MESSAGES

As the responder generates a response to each request, it shall identify the request with which the response is associated by inserting a PSN in the BTH of the response.

This allows the requester to correlate response packets it receives with its request. This basic concept is illustrated below in Figure 90 Example: PSNs for Response Messages on page 307.

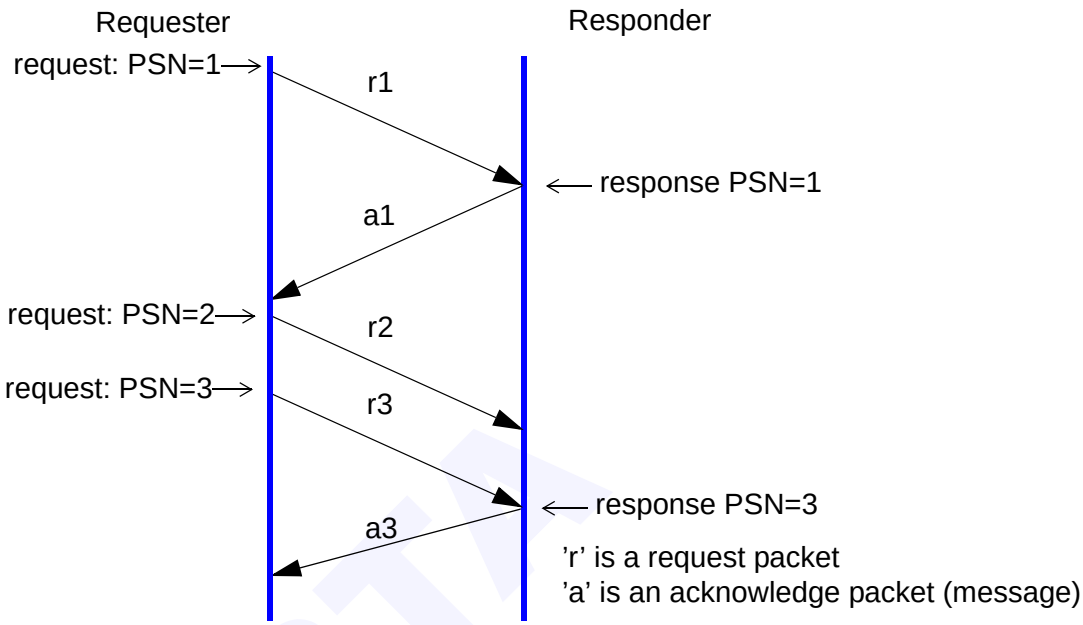


Figure 90 Example: PSNs for Response Messages

C9-95: For responses to SEND requests or RDMA WRITE requests the responder shall insert in the PSN field of the response the PSN of the most recent request packet being acknowledged.

"PSN of the most recent request", as used here and throughout Chapter 9, specifically means the PSN of the most recently received NEW request packet. (This distinguishes it from the PSN of a recently received duplicate request). Thus, the PSN of the most recently received request marks the point of greatest forward progress, as perceived by the responder, while ignoring duplicate requests.

Because of the rules for coalescing acknowledges (given in Section 9.7.5.1.2), the PSNs for consecutive response packets may not necessarily be sequential.

C9-96: For HCA responses to RDMA READ requests, the PSNs of the response packets must be sequential and monotonically increasing beginning with the PSN of the original RDMA READ request message.

o9-57: If a TCA implements RDMA READ functionality, then for each RDMA READ response the PSNs of the response packets must be se-

quential and monotonically increasing beginning with the PSN of the original RDMA READ request message.

o9-58: Since ATOMIC Operation requests require an explicit response, and since an ATOMIC Operation request is restricted to a single packet, the PSN of the response packet must be identical to the PSN of the request.

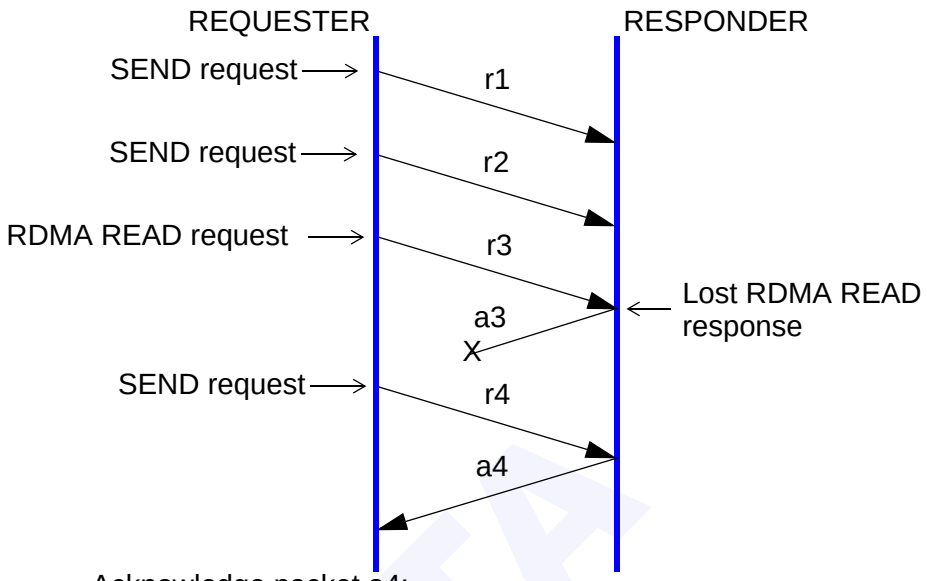
9.7.5.1.2 COALESCED ACKNOWLEDGE MESSAGES

It is not required that there be a unique, discrete response for each request packet. Instead, the responder may acknowledge several outstanding request packets with a single acknowledge packet. This is called acknowledge coalescing.

A given response packet acknowledges prior outstanding requests (i.e., those with earlier PSNs than the PSN contained in the BTH of the response packet) as follows:

1) If there is an outstanding RDMA READ or ATOMIC Operation request with a PSN earlier than the PSN in the BTH of the response packet, then the response packet implies a negative acknowledgment for the oldest such outstanding RDMA READ or ATOMIC Operation request. Any requests posted to the send queue subsequent to such an RDMA READ or ATOMIC Operation request are not acknowledged. This is illustrated in Figure 91 Requester Interpretation of Coalesced Acknowledges on page 309.
2) It implies a positive acknowledgment for all outstanding SEND or RDMA WRITE request packets with a PSN earlier than the PSN in the BTH of the response packet, unless such an outstanding SEND or RDMA WRITE request falls after an outstanding RDMA READ or ATOMIC Operation request as described above.
3) If the given response is an RDMA READ response message, it is the first (or only) packet of a RDMA READ response message that implicitly acknowledges prior outstanding requests.
4) The last (or only) packet of a RDMA READ response message explicitly acknowledges only the RDMA READ request.

These rules are illustrated in Figure 92.



Acknowledge packet a4:

1) implicitly acknowledges SEND requests r1 and r2,
2) implicitly NAKs RDMA READ request r3.
3) does not acknowledge SEND request r4.

Thus, acknowledges for requests r1 and r2 have been coalesced, and the requester must re-try requests r3 and r4.

Figure 91 Requester Interpretation of Coalesced Acknowledges

9.7.5.1.3 ACKNOWLEDGING RDMA READ REQUESTS

An RDMA READ response is different from a normal response in that it contains a data payload.

Every RDMA READ request message requires a discrete acknowledgment, called the RDMA READ response which consists of one or more packets.

C9-97: For an HCA, if an RDMA READ response contains more than one packet, the first and last packets must contain an AETH.

o9-59: In a TCA implementing RDMA, if an RDMA READ response contains more than one packet, the first and last packets must contain an AETH.

The AETH in the first packet implicitly acknowledges prior outstanding requests as specified in Section 9.7.5.1.2 Coalesced Acknowledge Messages on page 308. The AETH in the last packet acknowledges the RDMA READ request.

C9-98: For an HCA, if an RDMA READ response is itself a single packet, then that packet must contain an AETH.

o9-60: If a TCA implements RDMA functionality, and an RDMA READ response is itself a single packet, then that packet must contain an AETH.

The AETH contained in a single packet RDMA READ response serves to both implicitly acknowledge prior outstanding requests as well as to explicitly acknowledge the RDMA READ request itself.

C9-99: An HCA responder shall generate RDMA READ response packet payload lengths which are consistent with the opcode as follows:

1) A packet with an opcode of “RDMA READ response only” shall be zero to (PMTU) bytes long.
2) A packet with an opcode of “RDMA READ response first” or RDMA READ response middle” shall be exactly (PMTU) bytes long.
3) A packet with an opcode of “RDMA READ response last” shall be one to (PMTU) bytes long.
4) Zero length RDMA READ requests are permitted.
5) A response to a zero length RDMA READ request shall consist of a single packet with an opcode of “RDMA READ response only”.

o9-61: If a TCA implements RDMA functionality, it shall generate response packets with payload lengths as described in the previous compliance statement.

C9-100: If an HCA responder detects an error while in the process of returning a multi-packet RDMA READ response, it shall force a premature termination of the RDMA READ response by not transmitting any of the errant payload data and forcing the opcode of the packet on which the error occurred to “acknowledge” instead of an opcode of “RDMA READ response last”. The appropriate NAK code is inserted.

o9-62: If a TCA implements RDMA functionality, and detects an error while in the process of returning a multi-packet RDMA READ response, it shall force a premature termination of the RDMA READ response by not transmitting any of the errant payload data and forcing the opcode of the packet on which the error occurred to “acknowledge” instead of an opcode of “RDMA READ response last”. The appropriate NAK code is inserted.

Due to the relaxed ordering rules for RDMA READ Requests, the responder is permitted to begin executing one or more SEND or RDMA WRITE requests that arrive after the RDMA READ request.

C9-101: For an HCA, before executing any of the requests following the RDMA READ request, the header fields of the RDMA READ request must be validated. These requests must not be acknowledged until the outstanding RDMA READ responses have been sent.

o9-63: Before executing any of the requests following the RDMA READ request, the header fields of the RDMA READ request must be validated. These requests must not be acknowledged until the outstanding RDMA READ responses have been sent.

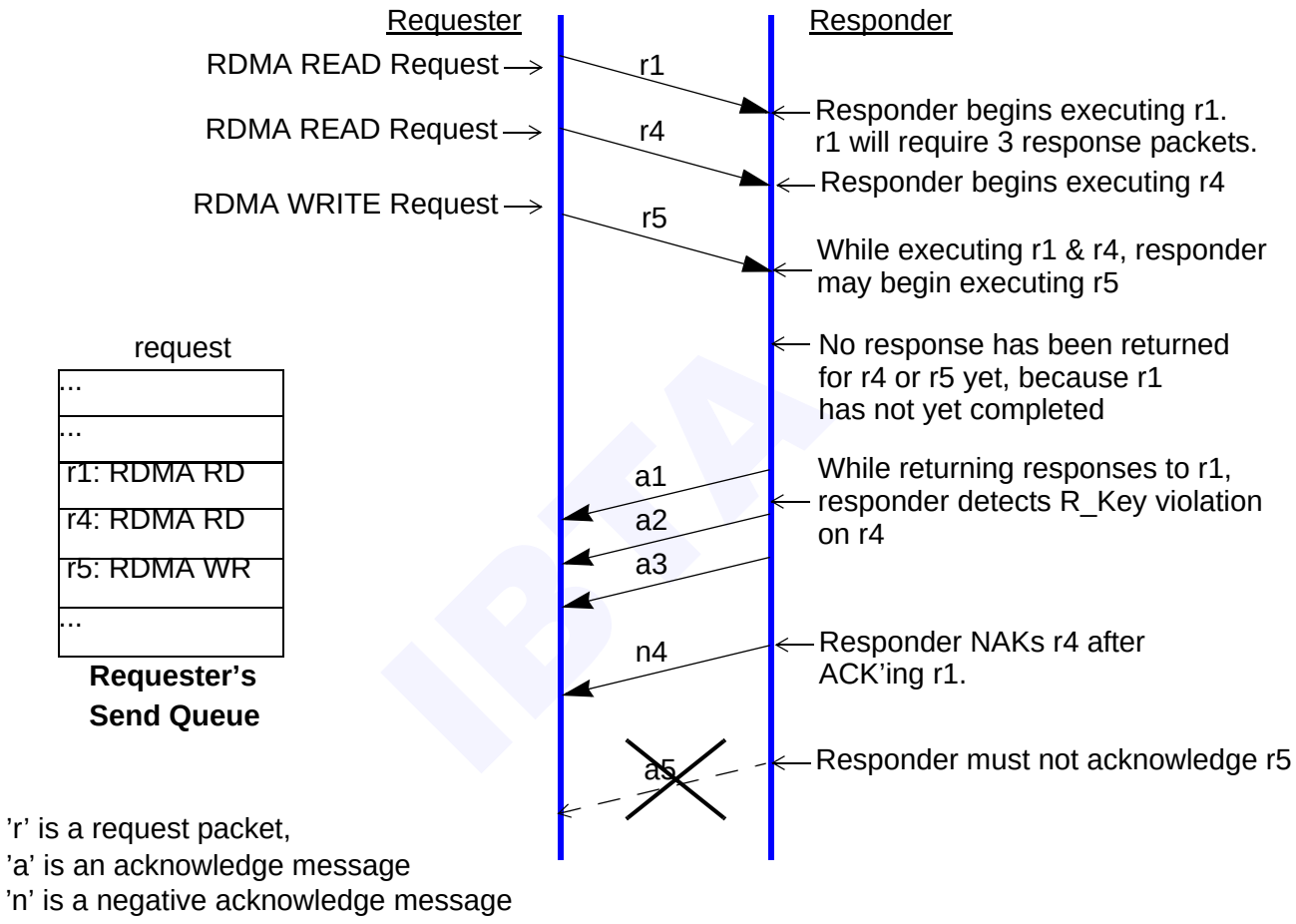


Figure 92 Relaxed Ordering Rules for RDMA READs

9.7.5.1.4 ACKNOWLEDGING DUPLICATE REQUESTS

After validating a duplicate request, the response to a duplicate request packet is as follows:

C9-102: After validating a duplicate request, if the duplicate packet is valid, the responder shall generate a response.

C9-103: Throughout the processing of the duplicate request, the responder shall not update its expected PSN; it remains set to the value it

had prior to the arrival of the duplicate request. This is true even if the responder detects an error while in the process of generating the response to the duplicate request.

Following generation of the appropriate response (as described in the next paragraphs), the responder resumes waiting for a new inbound packet with a PSN matching its expected PSN.

It is possible that the responder will receive another duplicate request while waiting for a new inbound packet. This is perfectly valid, and should be treated as simply another duplicate request. Furthermore, since it is a duplicate request, there is no requirement that the next request received be in sequential PSN order with the first duplicate request. However, the responder is required to maintain the same ordering rules for generating responses to duplicate requests as are required for normal new requests.

C9-104: In particular, a duplicate RDMA READ or ATOMIC Operation request must be acknowledged with an explicit response prior to returning acknowledges for subsequent duplicate SEND or RDMA WRITE requests.

This is illustrated in Figure 93 Maintaining the Order of Responses to Duplicate Requests on page 314.

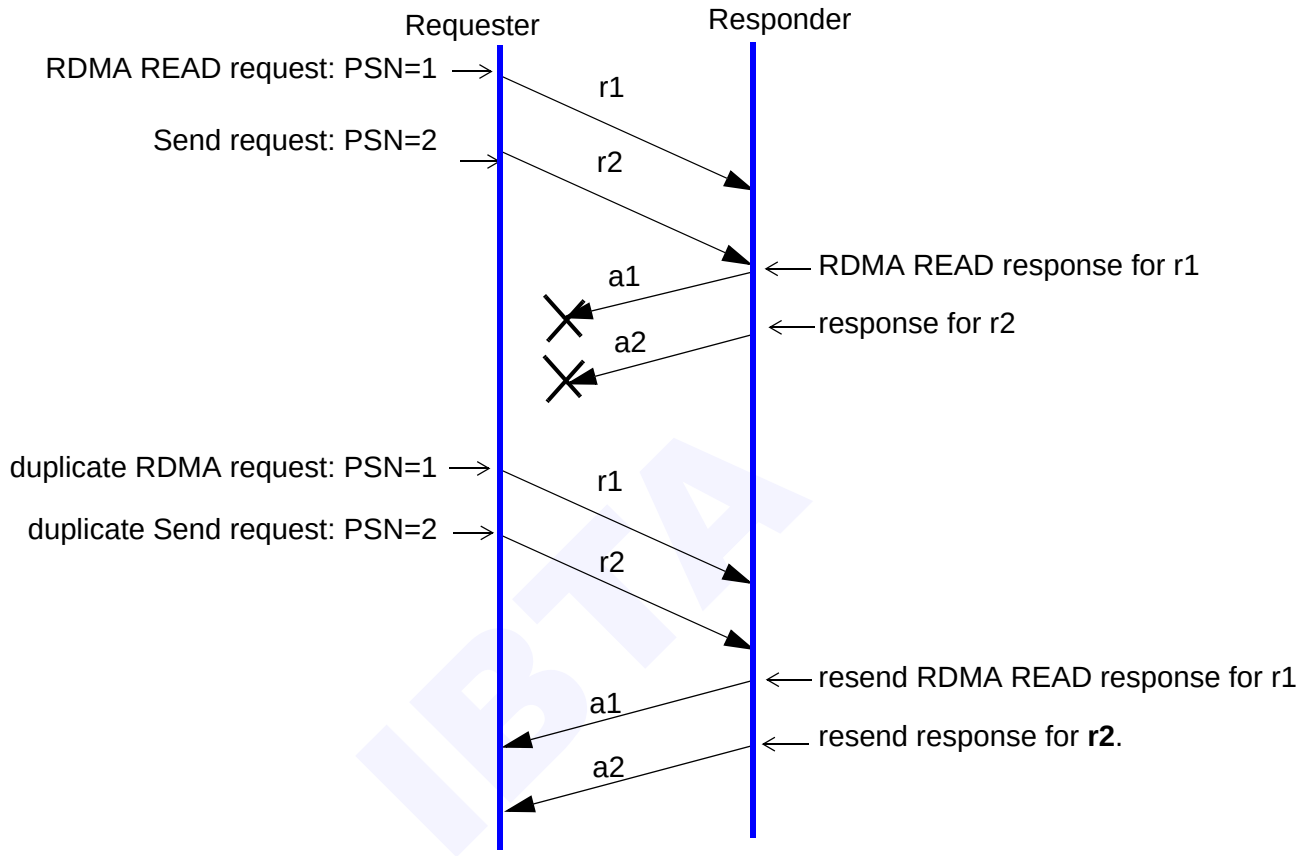


Figure 93 Maintaining the Order of Responses to Duplicate Requests

The response to be generated is a function of the duplicate request message as follows:

- SEND, RESYNC or RDMA WRITE Request

C9-105: For an HCA which receives a duplicate inbound SEND or RDMA Write request, or for a TCA which receives a duplicate inbound SEND request, the responder shall not re-execute the request but only generates a response packet for the duplicate packet, pending responses for any

outstanding duplicate RDMA READ requests or ATOMIC Operation requests.

o9-64: If a TCA responder implements RDMA functionality, it shall not re-execute the RDMA WRITE request but only generate a response packet for the duplicate packet, pending responses for any outstanding duplicate RDMA READ requests or ATOMIC Operation requests.

> The PSN of the acknowledge message must be the PSN of the most recently completed new request. One way to think of this process is as a logical extension of the ability to coalesce acknowledges. Indeed, the requester, on receiving a response to a duplicate request, treats it exactly as it would any other coalesced acknowledge; any outstanding duplicate RDMA READ or ATOMIC Operation requests are considered to be NAK'ed. In this case, by returning the PSN of the most recently completed request, the responder is informing the requester that it believes it has already seen and executed all requests between the duplicate request and the most recently completed request. This is illustrated in Figure 94.

C9-106: For duplicate SEND, RESYNC or RDMA WRITE requests, if the responder detects an error while in the process of returning the response, it shall silently drop the duplicate request. This is done in order to avoid confusion with any possible outstanding NAKs to new requests.

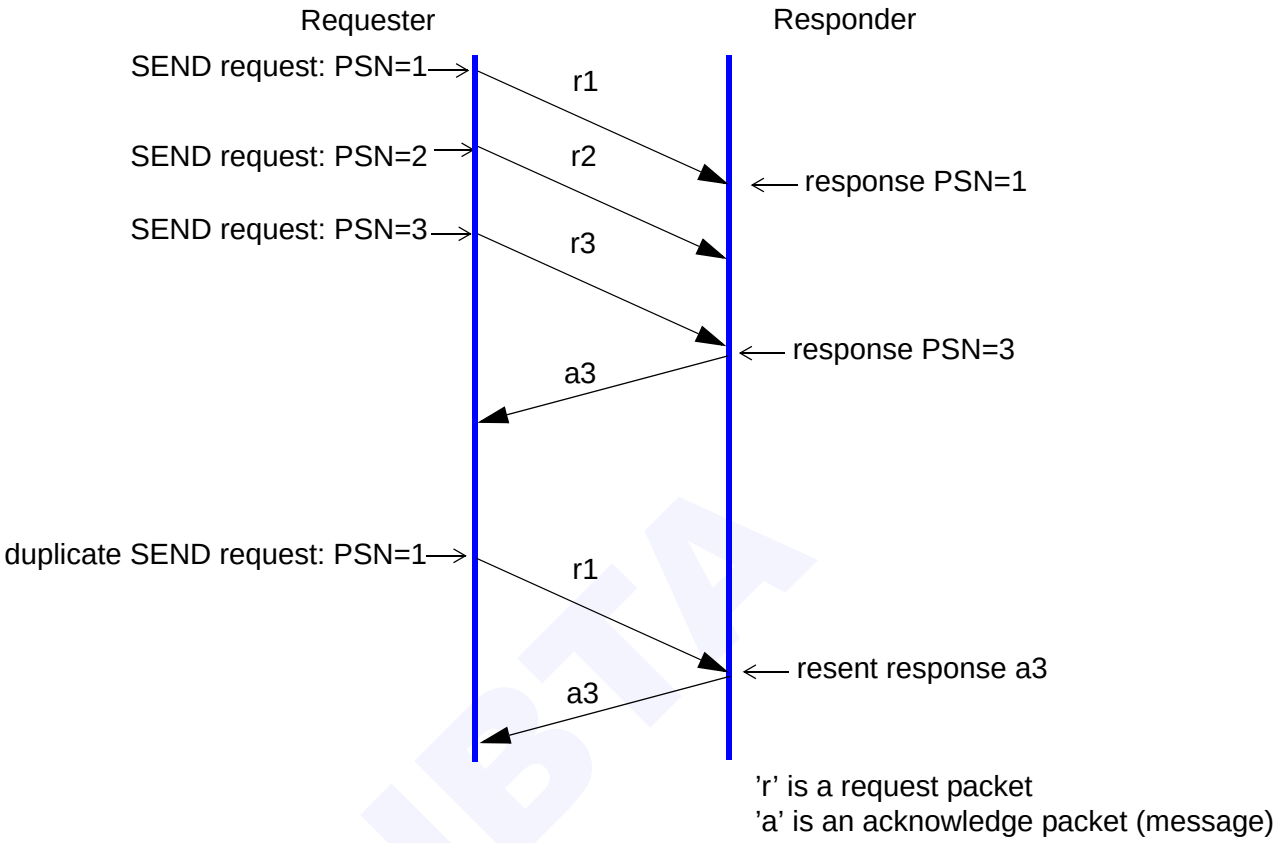


Figure 94 Acknowledging a Duplicate SEND Request

- RDMA READ Request

C9-107: An HCA responder must re-create the requested read response data. The resulting read data is returned to the requester in an RDMA READ response. The PSN of the first RDMA READ response packet shall be the same as the PSN of the duplicate request, with the PSNs for the subsequent response packets incrementing according to the normal rules for generating PSNs for RDMA READs.

C9-108: If an HCA responder detects an error while re-executing a duplicate RDMA READ request before returning the first response packet, the responder shall silently drop the duplicate request.

C9-109: If an HCA responder detects an error while re-executing a duplicate RDMA READ Request after returning one or more response packets, the RDMA READ response operation shall be aborted, i.e. no more response packets shall be returned.

When an RDMA READ Request is generated, a certain number of sequential PSN numbers are allocated based on the number of packets expected in the RDMA READ Response. These PSNs are used by the responder when generating the RDMA READ Response packet(s) Also, the original request contains a DMA Length defined in the RETH which represents the extent of the data being requested.

As described in Section 9.4.4 RDMA READ Operation on page 256, to be considered a duplicate RDMA READ Request, the PSN of the duplicate request must be within the responder's current duplicate PSN region. Furthermore, to be considered a valid duplicate RDMA READ Request, the PSN of the duplicate request must fall within the range of PSNs allocated to the original RDMA READ Response, and the amount of data requested in the duplicate request must be entirely contained within the extent of data requested in the original RDMA READ Request. In other words, the data requested in the duplicate RDMA READ Request must be a proper subset of the data requested in the original RDMA READ Request.

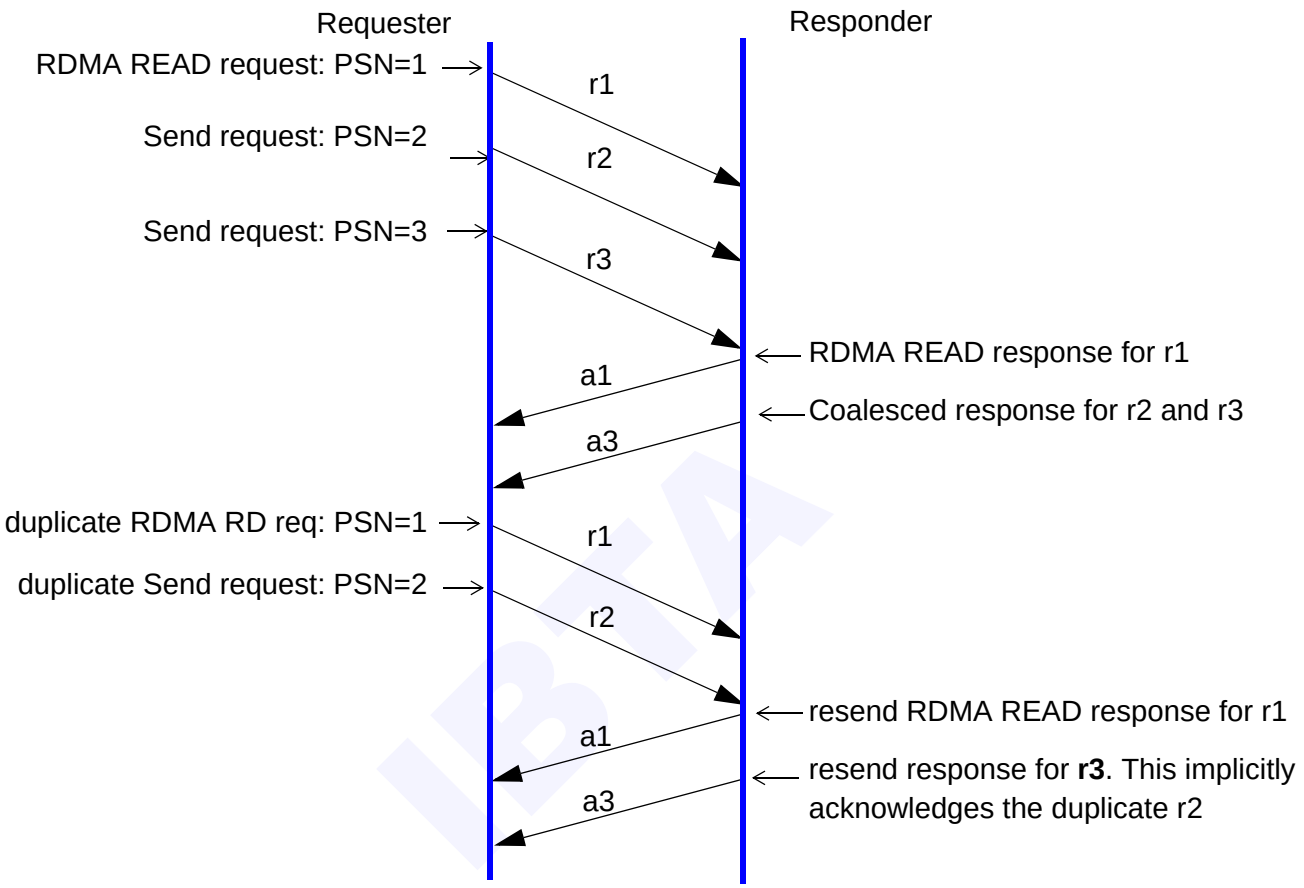
If the starting PSN and length of a duplicate RDMA READ Request does not fall within the range of PSNs allocated to the original RDMA READ Response, the request is invalid and the responder may silently drop the duplicate RDMA READ Request.

C9-110: A responder shall respond to all duplicate requests in PSN order; i.e. the request with the (logically) earliest PSN shall be executed first. If, while responding to a new or duplicate request, a duplicate request is received with a logically earlier PSN, the responder shall cease responding to the original request and shall begin responding to the duplicate request with the logically earlier PSN.

If a responder is interrupted by a duplicate request, it is not required to resume the interrupted request. It is the requester's responsibility to resend any unacknowledged requests.

o9-65: If a TCA implements RDMA functionality, RDMA READ Responses shall conform to the previous 4 compliance statements for HCAs.

> Following the duplicate RDMA READ response, the responder may acknowledge any subsequent duplicate Send or RDMA WRITE requests with the PSN of the most recently completed request. This is illustrated in Figure 95



'r' is a request packet
'a' is an acknowledge packet (message)

Figure 95 Acknowledging a Duplicate RDMA READ Request

- ATOMIC Operation Request

  A given receive queue may have resources to support only a limited number of ATOMIC Operations. When a duplicate ATOMIC Operation request is received, the PSN of the duplicate request is compared to the PSNs of the recently executed ATOMIC Operations.

o9-66: If the PSN of the duplicate ATOMIC Operation request matches exactly the PSN of one of the recently executed ATOMIC Operations, the saved results of that operation shall be returned to the requester. The responder shall not re-execute the request.

o9-67: If the PSN of the duplicate ATOMIC Operation request does not match the PSN of one of the recently executed ATOMIC Operations, the request is invalid and the duplicate request packet shall be silently dropped. This should never happen as long as the requester is observing the limits on the number of outstanding ATOMIC Operation requests.

o9-68: If a local error prevents the responder from reproducing the original ATOMIC Operation request data, the responder must silently drop the duplicate request.

In all cases, the PSN returned in the acknowledge message is the PSN of the duplicate request.

9.7.5.1.5 GENERATING NAKS

There are several circumstances that cause a responder to generate a NAK.

C9-111: In all cases except for RDMA READ requests, the PSN of the NAK packet shall contain the responder's expected PSN.

C9-112: In the case of an RDMA READ response packet, the PSN given in the NAK response packet shall point to the RDMA READ response packet which is being NAK'ed.

C9-113: When generating an RNR NAK, the PSN of the response packet shall point to the PSN of the packet being RNR NAK'ed.

C9-113.a1: When generating a NAK, the packet containing the NAK code shall have an opcode of Acknowledge.

This means that, even for an RDMA Read response, if the responder is returning a NAK code, it does so by substituting a packet with an opcode of Acknowledge instead of the normal opcode of RDMA Read Response (first/middle/last/only).

Once the responder has returned a NAK-sequence error or an RNR NAK, it waits for the requester to send a packet with the responder's expected PSN.

The rules that the responder must follow are as follows:

C9-114: Once a NAK has been returned for a PSN sequence error, the responder shall ignore all other new requests, except duplicate requests, until it receives a valid request with a PSN that matches its expected PSN. It shall not return any other NAK packets, except in response to a valid request with a PSN that matches its expected PSN.

C9-115: The responder must continue to respond to duplicate requests as specified above. However, the responder shall not return a NAK in response to an error condition occurring while processing a duplicate request.

9.7.5.1.6 ACKNOWLEDGE MESSAGE SCHEDULING

The scheduling of responses, per se, is not specified; however the requester may use the AckReq bit in the BTH to require the responder to schedule a response.

C9-116: When the responder receives a valid request packet with the AckReq bit set, it shall schedule a response packet for that request. The PSN of the response must be equal to or logically greater than (modulo 2^{24}) the PSN of the request.

After receiving a request message with the AckReq bit set, the responder still accepts request messages, (including additional ones marked with the AckReq bit set), while it is preparing to transmit the response packet. These additional request messages may result in a coalesced ACK when the responder is able to send the response. In this case, the single response message may satisfy several outstanding request messages.

When one or more requests arrive without the AckReq bit set, the responder may choose to deliberately coalesce its responses; it may even wait an unbounded time for additional requests, until one arrives that requires the scheduling of a response.

There are several places where the AckReq bit can be very useful to the requester. For example, if the requester is sending the last packet of the last request WQE posted to the send queue, it is advisable for the requester to set the AckReq bit or use some other mechanism to force the responder to return a response. If the requester does not do so, there is a possibility that the responder will choose to coalesce responses indefinitely. Some other mechanisms that the requester can use to ensure that the responder returns a response are:

- Always set the AckReq bit on the last (or only) packet of every message
- Follow a given request with a NOP with the AckReq bit set
- Retry the request for which a response was desired with the AckReq bit set.

For SEND or RDMA WRITE requests, an ACK may be scheduled before data is actually written into the responder's memory. The ACK simply indicates that the data has successfully reached the fault domain of the responding node. That is, the data has been received by the channel adapter and the channel adapter will write that data to the memory system

of the responding node, or the responding application will at least be informed of the failure.

The absence of the AckReq bit does not prohibit the responder from generating a response packet. As always, RDMA READ and ATOMIC Operation requests require explicit responses, thus the AckReq bit has no effect on requests.

9.7.5.1.7 RESPONSE FORMATS

Responses may take one of three forms:

1) An acknowledge packet for a normal SEND, RESYNC or RDMA WRITE operations,
2) RDMA READ responses, and
3) Acknowledge messages for ATOMIC Operations - see 9.4.5 ATOMIC Operations on page 260.

The key distinctions between the three forms is that the normal acknowledge packet (used for SENDs, RESYNCs and RDMA WRITEs) does not carry a payload field, while the responses for both the RDMA READ and ATOMIC Operations do. This observation impacts both the format of the response and the rules for coalescing acknowledges.

An acknowledge packet contains the following information:

- A syndrome used to notify the requester of the success or failure of a given request message,
- A PSN value used by the requester to correlate the acknowledge message with its listing of outstanding requests,
- A Message Sequence Number used by the responder to notify the requester that request messages have been completed,
- Optional End-to-End flow control credits,
- Payload data in the case of a RDMA READ response or ATOMIC Operation response.

Each of the three forms is discussed in the following sections.

9.7.5.1.8 RESPONSE FORMAT FOR SEND, RESYNC OR RDMA WRITE REQUESTS

This format is used to acknowledge SEND, RESYNC or RDMA WRITE request packets. A normal acknowledge message comprises a single packet, and is shown in Figure 96.

LRH	GRH	BTH	RDETH	AETH	ICRC	VCRC

note 1: GRH may or may not appear, depending on the LRH Next Header field
note 2: RDETH appears only for reliable datagram operations
note 3: DETH, RETH, EOP, PYLD and IMM fields are prohibited

Figure 96 Response Format for SENDs, RDMA WRITEs

9.7.5.1.9 RDMA READ RESPONSES

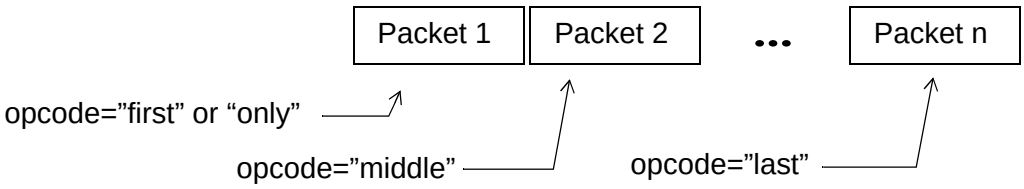
This response format, called a RDMA READ response, is used to acknowledge RDMA READ requests. A RDMA READ response message consists of one or more packets.

C9-117: For an HCA, the PSNs of the RDMA READ response packets must be sequential and monotonically increasing. If the response message consists of more than one packet, the first and last packets of the response message must contain an Acknowledge Extended Transport Header (AETH).

o9-69: If a TCA implements RDMA functionality, the PSNs of the RDMA READ response packets must be sequential and monotonically increasing. If the response message consists of more than one packet, the first and last packets of the response message must contain an Acknowledge Extended Transport Header (AETH).

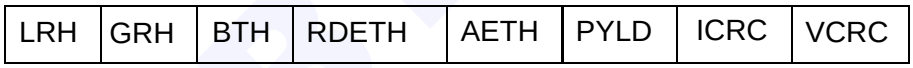
C9-118: For an HCA, if the response message contains only a single packet (an “only” packet), then that packet must contain an AETH. This is illustrated in Figure 97.

o9-70: If a TCA implements RDMA functionality, and the response message contains only a single packet (an “only” packet), then that packet must contain an AETH as shown in Figure 97.



Each RDMA READ Response message comprises one or more pack-

Format for all middle packets. PYLD must be (PMTU) bytes long.



Format for first, last or only RDMA READ Response Packet.

If a first packet, PYLD shall be (PMTU) bytes long.
If an only packet, PYLD shall be zero to (PMTU) bytes long.
If a last packet, PYLD shall be one to (PMTU) bytes long.

note 1: GRH may or may not appear, depending on the LRH:Next Header field
note 2: RDETH appears only for reliable datagram operations
note 3: DETH, RETH, and IMM headers are prohibited

Figure 97 Acknowledge Message Format for RDMA READ Requests

A RDMA READ Response message, besides acknowledging the RDMA READ request itself, also implicitly acknowledges requests preceding the RDMA READ request. The rules governing coalesced ACKs are given in Section 9.7.5.1.2 Coalesced Acknowledge Messages on page 308.

The arrival of either a first packet or an only packet triggers the implicit acknowledges of any outstanding request messages as specified in section 9.7.5.1.2 Coalesced Acknowledge Messages on page 308. This is done

in order to reduce the latency to complete any outstanding request messages.

The arrival of a last packet or an only packet triggers the explicit acknowledge of the RDMA READ request itself.

9.7.5.2 AETH FORMAT

Acknowledge syndromes are carried in the AETH of the acknowledge message. The table below illustrates the syndrome field of the AETH.

C9-119: When generating an AETH, a HCA responder implementing RC service shall encode the AETH Syndrome Field as shown in Table 43 AETH Syndrome Field on page 324.

o9-71: If a TCA responder implements RC service, or if a CA responder implements RD service, the responder shall encode the AETH Syndrome Field as shown in Table 43 AETH Syndrome Field on page 324.

Table 43 AETH Syndrome Field

bit 7	bits 6:5	bits 4:0	Definition
0	0 0	C CCCC	ACK (C CCCC = credit count)
0	0 1	T TTTT	RNR NAK (T TTTT = timer value)
0	1 0	X XXXX	reserved
0	1 1	N NNNN	NAK (N NNNN = NAK code)

C9-119.a1: For an HCA, the msb of the AETH Syndrome Field is reserved and shall be set to zero.

o9-71.a1: For a TCA which provides RC or RD service, the msb of the AETH Syndrome Field is reserved and shall be set to zero.

o9-72: If a CA implements Reliable Datagram service, the C CCCC bits are set to zero, since end to end credits are not defined for RD service.

The interpretation of bits [4:0] depends on the code contained in bits [6:5]. Bits [4:0] may contain a positive acknowledgment with or without end-to-end flow control credits (depending on whether the service is RC or RD), an RNR NAK timer value, a positive acknowledgment without end-to-end credits, or a NAK code.

C CCCC = encoded end-to-end flow control credits

T TTTT = RNR NAK Timer Field - see Table 45 Encoding for RNR NAK Timer Field on page 330

N NNNN = NAK Code - see Table 44 NAK Codes on page 325

Code 011 N NNNN (NAK) allows MSNs to be carried with NAK packets.

Acknowledge syndromes are carried in the AETH of the acknowledge message. The table below illustrates the syndrome field of the AETH.

9.7.5.2.1 END-TO-END FLOW CONTROL CREDIT FIELD

If bits [7:5] of the AETH Syndrome field are zero, then bits [4:0] of the AETH Syndrome field carries encoded end-to-end flow control credits from the responder to the requester. This field is only valid for reliable connections.The encoding 5b11111 means that the credit field is not valid. This encoding is also used for cases where the receive queue does not support End-to-End credits. See Section 9.7.7.2 End-to-End (Message Level) Flow Control on page 347 for further details.

9.7.5.2.2 NAK CODES

If bits [6:5] of the AETH Syndrome field are b11, then bits [4:0] carry a NAK code. The code guides the requester in selecting a recovery strategy. The following sections describe all the possible NAK Codes. Even though an RNR NAK has its own AETH syndrome (AETH[6:5] = b01), RNR NAK is also described in this section.

The list of valid NAK codes is provided in Table 44.

Table 44 NAK Codes

NAK Code (AETH bits 4:0)	Definition
0 0000	PSN Sequence Error
0 0001	Invalid Request
0 0010	Remote Access Error
0 0011	Remote Operational Error
0 0100	Invalid RD Request
0 0101 - 1 1111	reserved

C9-120: If a requester receives an acknowledge message containing a reserved code, it shall consider the acknowledge packet to be malformed and shall silently drop it. This may eventually cause the requester to time out while waiting for the missing acknowledge packet, at that time it will either re-transmit the original request message, or stop operations on that send queue.

9.7.5.2.3 PSN SEQUENCE ERROR

A sequence error occurs when a responder detects a packet that is out of PSN sequence, i.e. a PSN value that is neither equal to the expected PSN nor within the valid duplicate packet range.

C9-121: The responder, when it constructs NAK packet in response to a sequence error, must insert its expected PSN value in the PSN field of the BTH. This lets the requester back up its send queue to at least the point of the failure and begin re-sending request packets from that point forward.

A PSN sequence error may be retried by the requester a number of times. Once the retry count has expired, the requester's transport notifies its client that it did not succeed in transferring the message. The requester's required behavior once its retry count has expired is given in 9.9.2 Requester Side Error Behavior on page 397. The following discussion specifies the behavior before the retry count has expired.

When the responder detects a sequence error there is no impact on the receive queue nor are any WQEs consumed. Instead, the receive queue simply returns the NAK packet to the requester and resumes waiting for an inbound request packet with the correct PSN value.

C9-122: Once a NAK packet for a sequence error has been returned to the requester, the responder shall discard all subsequent requests that do not contain the responder's expected PSN, except for valid duplicate requests.

C9-123: If the responder receives a request packet with a PSN that is logically less than its expected PSN (i.e. a valid duplicate request packet), it shall respond to that request according to the rules for duplicate packet processing.

9.7.5.2.4 REMOTE ACCESS ERROR

A R_Key violation is caused by any or all of the following conditions for either a RDMA READ, RDMA WRITE, or ATOMIC Operation:

- The R_Key field of the RETH is invalid.
- The virtual address and length or type of access specified is outside the locally defined limits associated with the R_Key.
- For an HCA, a protection domain violation is detected.

C9-124: For an HCA responder, when reporting an RDMA remote access error, the BTH field of the acknowledge message must contain the PSN of the request packet that caused the remote access error.

o9-73: If a TCA responder implements RDMA functionality, or if a CA responder supports ATOMIC operations, then when reporting a remote access error, the BTH field of the acknowledge message must contain the PSN of the request packet that caused the remote access error.

The responder's behavior on detecting an access error, beside generating a NAK-Remote Access Error packet, is specified in section 9.9.3 Responder Side Behavior on page 408.

The requester's behavior on receiving a NAK-Remote Access Error is specified in section 9.9.2 Requester Side Error Behavior on page 397.

9.7.5.2.5 INVALID REQUEST

The requester has requested an operation that is outside the established usage of the transport service - generally, this is an OpCode that is not supported by the responder or a request whose length exceeds the available receive buffer space. For example, an RDMA request transmitted to a responder that does not support RDMAs would cause an Invalid Request Error. An out-of-sequence OpCode may also cause a NAK-Invalid Request depending on the particular service.

C9-125: When reporting an invalid request, the BTH field of the acknowledge packet must contain the responder's expected PSN value, i.e., the PSN of the request packet that contained the invalid request.

The responder's behavior upon detecting an invalid request, besides generating a NAK-Invalid Request, is given in section 9.9.3 Responder Side Behavior on page 408.

The requester's behavior on receiving a NAK-Invalid Request is given in section 9.9.2 Requester Side Error Behavior on page 397.

9.7.5.2.6 REMOTE OPERATIONAL ERROR

A remote operational error occurs when the responder encounters a situation that prevents its receive queue from completing the current request. The list of error conditions detectable by the responder, and reportable as a remote operational error, is not specified since it is implementation specific. Remote operational errors cannot be caused by anything the requester may have done. Rather, they reflect a fault in the responder.

C9-126: When reporting a remote operational error, the BTH field of the acknowledge message must contain the PSN of the request being executed at the time the responder detected the operational error.

The responder's behavior upon detecting an operational error, besides returning NAK-Remote Operational Error, is given in section 9.9.3 Responder Side Behavior on page 408.

The requester's behavior when it receives a NAK-Remote Operational Error is specified in section 9.9.2 Requester Side Error Behavior on page 397.

9.7.5.2.7 INVALID RD REQUEST

This NAK code is generated when the responder detects a Q_Key or RDD violation while operating in RD service, or if the destination QP is not configured for RD service, or if the destination QP is not in a state where it can accept an inbound packet.

o9-74: If the responder's EE Context detects an invalid P_Key, the request packet shall be silently dropped by the EE Context.

If no P_Key violation is detected, the EE Context forwards the packet to the receive queue specified in the BTH.

C9-127: If the QP as specified in the BTH is not configured for RD service, then a NAK-Invalid RD Request shall be returned.

The receive queue checks the Q_Key of the inbound request packet and also checks that its current RDD value matches that of the EE Context.

If the responder's receive queue detects an invalid Q_Key, or if the receive queue's RDD value does not match that of the EE Context, the responder shall return a NAK-Invalid RD Request to the requester.

The responder's behavior upon detecting either a Q_Key or RDD violation, beside generating a NAK-Invalid RD Request, is specified in section 9.9.3 Responder Side Behavior on page 408.

The requester's behavior in response to a NAK-Invalid RD Request is specified in section 9.9.2 Requester Side Error Behavior on page 397.

9.7.5.2.8 RNR NAK

Under certain circumstances, a receive queue may be temporarily unable to accept an inbound request message. For example, there may not currently be a valid receive WQE posted to the receive queue. When this occurs, the responder may return a response indicating Receiver Not Ready (RNR NAK). Note: the responder may return an RNR NAK for any type of request (e.g. SEND, RDMA READ request, RDMA WRITE request, etc.). On receiving a RNR NAK, the requester may, after waiting for at least the interval specified in the RNR NAK, retry the same request. "The same request" means the precise same request message beginning with the same PSN as reported by the responder in its RNR NAK packet.

C9-128: For an HCA requester using Reliable Connection service, after receiving an RNR NAK, the requester shall not substitute a different request message by reusing the same PSN.

o9-75: If a TCA requester implements Reliable Connection service, after receiving an RNR NAK, the requester shall not substitute a different request message by reusing the same PSN.

For Reliable Datagram service, the requester may either exactly repeat the request, move the EEC to the Error state, abandon the request or suspend the request as described in Section 9.7.8 Reliable Datagram on page 358.

C9-129: An HCA responder using Reliable Connection service, when generating an RNR NAK, shall indicate the appropriate interval in the timer field of the AETH. The value loaded in the timer field of the AETH shall be as shown in Table 45 Encoding for RNR NAK Timer Field on page 330.

o9-76: If a CA responder implements Reliable Datagram service, or if a TCA implements Reliable Connection service, it shall follow this rule: when generating an RNR NAK, the responder shall indicate the appropriate interval in the timer field of the AETH. The value loaded in the timer field of the AETH shall be as shown in Table 45 Encoding for RNR NAK Timer Field on page 330.

C9-130: An HCA requester providing Reliable Connection service, after receiving a RNR NAK, must wait for at least the interval specified in the timer field of the AETH before retrying the request. If the requester fails to wait for the appropriate timeout interval before re-trying the request, the responder may silently drop the packet.

o9-76.a1: An HCA requester providing Reliable Datagram service, after receiving a RNR NAK, must wait for at least the interval specified in the timer field of the AETH before retrying the request. If the requester fails to wait for the appropriate timeout interval before re-trying the request, the responder may silently drop the packet.

o9-76.a2: A TCA requester providing Reliable Connection service, or a TCA requester providing Reliable Datagram service, after receiving a RNR NAK, must wait for at least the interval specified in the timer field of the AETH before retrying the request. If the requester fails to wait for the appropriate timeout interval before re-trying the request, the responder may silently drop the packet.

C9-131: This compliance statement is obsolete and has been removed.

C9-132: An HCA requester using Reliable Connection service shall maintain a 3 bit retry counter which is loaded during connection establishment with information provided by the responder. This counter is used to limit the number of times a requester can retry an operation which was RNR NAK'ed. When a RNR NAK response is received, if the RNR NAK retry counter is not equal to 7 (indicates infinite retry), the requester shall decrement the RNR NAK retry counter. Thereafter, when the retry timer expires, if the retry counter is non-zero, the requester may re-issue the request.

o9-77: If a CA requester implements Reliable Datagram service, or if a TCA requester implements Reliable Connection Service, it shall maintain a 3 bit retry counter which is loaded during connection establishment with information provided by the responder. This counter is used to limit the number of times a requester can retry an operation which was RNR NAK'ed. When a RNR NAK response is received, if the RNR NAK retry counter is not equal to 7 (indicates infinite retry), the requester shall decrement the RNR NAK retry counter. Thereafter, when the retry timer expires, if the retry counter is non-zero, the requester may re-issue the request.

A locally detected error is recorded by the requester if the retry counter has decremented to zero at the time that the RNR NAK retry timer expires. See Section 9.9.2.1 Requester Side Error Detection - Locally Detected Errors on page 397 for further details.

The timer field is encoded as shown in the Table below.

Table 45 Encoding for RNR NAK Timer Field

RNR Time	Delay in milliseconds	RNR Time	Delay in milliseconds
00000	655.36	10000	2.56
00001	0.01	10001	3.84
00010	0.02	10010	5.12
00011	0.03	10011	7.68
00100	0.04	10100	10.24
00101	0.06	10101	15.36

Table 45 Encoding for RNR NAK Timer Field (Continued)

RNR Time	Delay in milliseconds	RNR Time	Delay in milliseconds
00110	0.08	10110	20.48
00111	0.12	10111	30.72
01000	0.16	11000	40.96
01001	0.24	11001	61.44
01010	0.32	11010	81.92
01011	0.48	11011	122.88
01100	0.64	11100	163.84
01101	0.96	11101	245.76
01110	1.28	11110	327.68
01111	1.92	11111	491.52

The use of RNR NAK for temporary problems that do not affect the whole message (such as a memory page not present) is not prohibited. In particular, for Reliable Datagram service, an RNR NAK returned in the middle of a SEND request message by a responder may result in the current message being abandoned by the requester and a new message being sent from another queue pair. This may result in unexpected incomplete messages at the responder. These incomplete messages are detected by the responder while executing a RESYNC request, thus allowing the responder to complete the partially completed WQE in error and begin receiving the new request.

A responder should use this feature as a mechanism to delay the incoming request when a local resource is unavailable only rarely. The RNR NAK mechanism consumes bandwidth in that an incoming packet will be aborted and will have to be re-sent.

9.7.6 REQUESTER: RECEIVING RESPONSES

9.7.6.1 VALIDATING INBOUND RESPONSE PACKETS

On receipt of an inbound acknowledge packet, a requester validates the packet as follows:

C9-133: To verify the integrity of the packet, the requester shall validate the packet as specified in Section 9.6 Packet Transport Header Validation on page 269. Invalid packets shall be silently dropped by the requester.

C9-134: For an HCA requester using Reliable Connection service, the PSN shall be examined to detect out of order packets. Since acknowledges may be coalesced as described in section 9.7.5.1.2 Coalesced Acknowledge Messages on page 308, the PSN is used to detect coalesced responses.

o9-78: If a TCA requester implements Reliable Connection service, or if a CA requester implements Reliable Datagram service, the PSN of each acknowledge packet shall be examined to detect out of order packets. Since acknowledges may be coalesced as described in section 9.7.5.1.2 Coalesced Acknowledge Messages on page 308, the PSN is used to detect coalesced responses.

C9-135: For an HCA requester using Reliable Connection service, the validity of the acknowledge syndrome shall be checked according to the table in Section 9.7.5.2.2 NAK Codes on page 325. A response packet containing a reserved NAK code shall be simply dropped.

o9-79: If a TCA requester implements Reliable Connection service, or if a CA requester implements Reliable Datagram service, the validity of the acknowledge syndrome shall be checked according to the table in Section 9.7.5.2.2 NAK Codes on page 325. A response packet containing a reserved NAK code shall be simply dropped.

o9-79.a1: If a CA implements Reliable Datagram service, when receiving a response packet, the requester shall check the destination QPn contained in the BTH against the expected QPn for the current EEC. If the response packet is not destined for the currently active requester side QP, it shall be dropped by the requester.

The requester must also insure that responses are appropriate for the type of operation being performed. For example, in certain congested fabric cases it is possible for an ACK to arrive when an RDMA READ or Atomic response is expected. This can happen even when the PSN contained in the response packet matches the requester's expected response PSN. The converse is also true.

Therefore, the requester should validate that the response packet received is consistent with the outstanding request. If it is not, the requester's behavior should be as described in Table 56 Requester Side Error Behavior on page 401.

C9-135.a1: This compliance statement is obsolete.

If the packet is determined to be valid, it is processed by the requester. While processing the acknowledge packet, the requester may encounter local errors. The list of local errors that the requester may encounter when processing the acknowledge message is not specified since it is implementation specific, but includes any error due to a fault on the requester side. The required behaviors for this case are specified in 9.9.2 Requester Side Error Behavior on page 397.

Validating the PSN of an inbound response packet relies on identifying three critical points in the PSN sequence. These three points are:

1) Requester's maximum forward progress - the logically largest (modulo 2^{24}) PSN of any request sent by the requester. (This includes PSN space allocated for RDMA READ responses.) It marks the "right-hand" edge of the Valid Region.
2) Oldest unacknowledged request - the PSN of the oldest outstanding (unacknowledged) request. This represents the maximum forward progress made by the responder, as viewed by the requester. The significance of this PSN is that it marks the end of the duplicate region, i.e. responses with PSNs logically less than this are treated either as invalid or as duplicates.
3) Oldest valid request - this point marks the "left-hand" edge of the Valid Region.

See the figure below for an illustration of these three points.

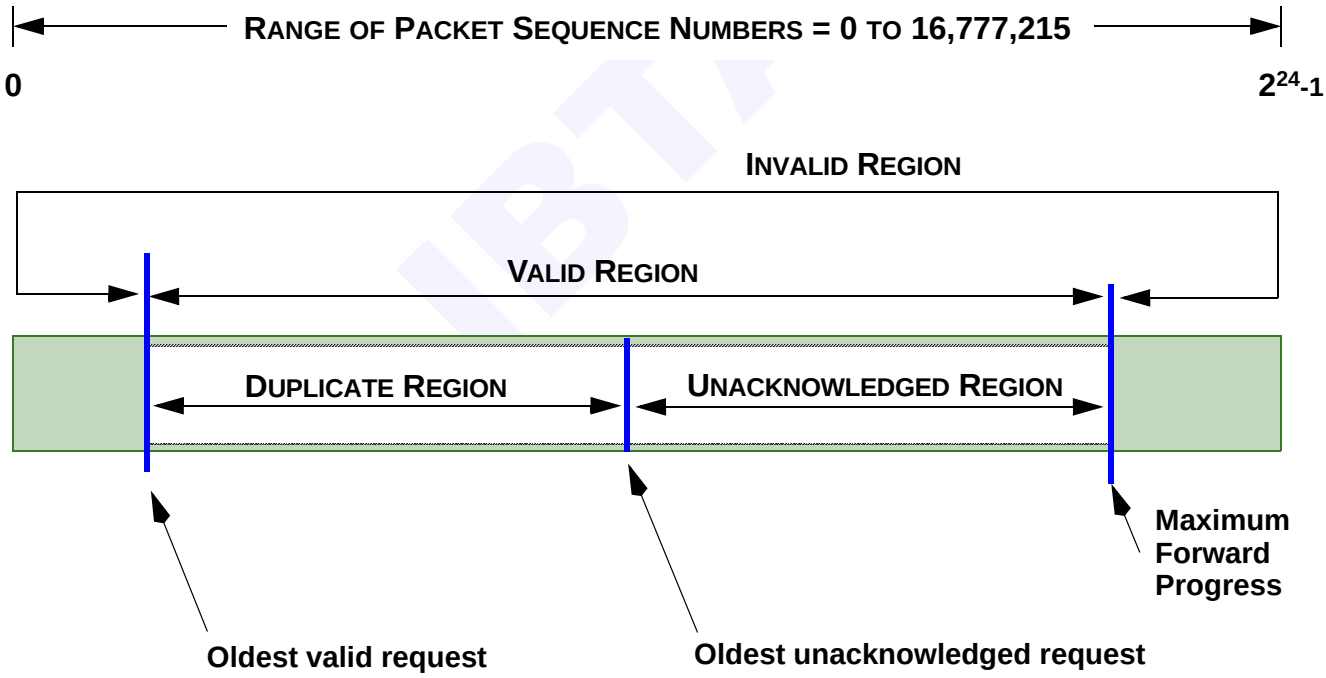


Figure 98 Response Packet PSN Regions

As is the case with request packets, each response packet carries a PSN. The requester, on receiving a response packet, checks the PSN to determine if the response is an expected response or a ghost acknowledge packet. Conceptually, the requester keeps track of the PSN of the oldest unacknowledged request packet and the logically largest (modulo 2^{24}) PSN sent to date including PSNs reserved for RDMA READ responses.

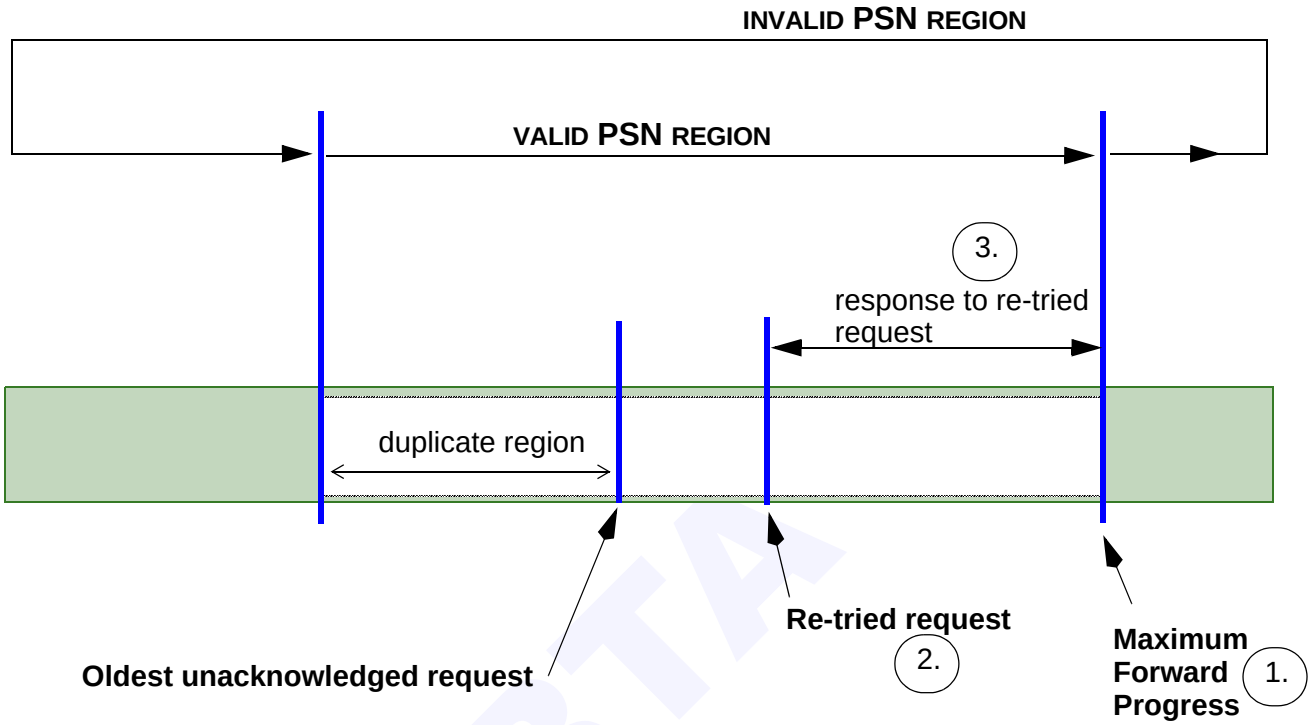
These two PSNs define the endpoints of a range of PSNs identified in the figure above as the Unacknowledged Region. If the PSN of a response packet falls within that range then the packet is an expected response packet. If the response does not fall within that region, then it is considered either a duplicate response and is handled according to the rules defined in Section 9.7.5.1.4 Acknowledging Duplicate Requests on page 312, or it a ghost (invalid) acknowledge packet and is dropped by the requester.

C9-136: For an HCA requester using Reliable Connection service, ghost acknowledge packets shall be dropped by the requester.

o9-80: If a TCA requester implements Reliable Connection service, or if a CA requester implements Reliable Datagram service, ghost acknowledge packets shall be dropped by the requester.

9.7.6.1.1 PSNs FOR RETRIED REQUESTS

Under some circumstances (described below) the requester may need to retry a request. In PSN terms, this means that the requester may re-send a packet with a PSN which is logically less (modulo 2^{24}) than the maximum PSN transmitted to date. In the figure below, this is identified as the Re-tried request.



1. Requester advances to the point of maximum forward progress.

2. Due to timeout, the request "backs up" and re-tries a request.

3. Meanwhile, the responder may have continued to make progress. Thus, the response it returns may have a PSN logically greater than the PSN of the re-tried request. (However, it cannot be greater than the requester's point of maximum forward progress.)

Figure 99 Three PSN Paradigm

During the process of retrying a request, the requester should maintain a means of marking the point of furthest PSN advance (high water mark) even though it is logically "backing up" the PSN sequence when it re-tries a request. This is necessary because the response to the re-tried request may have a PSN which is logically greater than the PSN of the re-tried request. This can happen because under some circumstances the responder might interpret the re-tried request as a duplicate request. If so, it (the responder) is obligated to return the PSN marking its furthest point of advance, which may be beyond the PSN of the re-tried request.

The need for the requester to maintain these three pointers (point of maximum advance, PSN of the oldest unacknowledged request, and the PSN of the re-tried request) is referred to as the three PSN paradigm. Although various implementations may find different ways of implementing the three PSN paradigm, nonetheless, a requester must account for the fact that a responder may return a response to a re-tried request with a PSN which is logically greater (further advanced) than the PSN of the re-tried request itself.

9.7.6.1.2 REQUESTER RESPONSE TO A NAK MESSAGE

The requester's reaction to a negative response message depends on the NAK code that is returned, and whether the queue pair is configured for reliable connected or reliable datagram service.

A NAK-Sequence error triggers an automatic retry of the request. The PSN in the response packet is the requester's indication of the request packet that the responder believes it missed, thus, the requester can retry that request. To prevent the requester from retrying the same request forever, the requester maintains a 3 bit retry counter which is used to count the number of times a particular request packet has been retried and timed out. See Section 9.9.2.1 Requester Side Error Detection - Locally Detected Errors on page 397 for a full description of the retry counter.

C9-137: An HCA requester using Reliable Connection service shall decrement its 3 bit retry counter each time the responder returns a NAK-Sequence error for a given request packet. The counter shall be re-loaded whenever the given outstanding request is cleared. If automatic path migration is not supported, and if a NAK-Sequence error is returned once more, then the requester shall declare a locally detected error.

o9-80.a1: An HCA requester which supports Reliable Datagram service shall decrement its 3 bit retry counter each time the responder returns a NAK-Sequence error for a given request packet. The counter shall be re-loaded whenever the given outstanding request is cleared. If automatic path migration is not supported, and if a NAK-Sequence error is returned once more, then the requester shall declare a locally detected error.

o9-81: A TCA requester which implements Reliable Connection service or Reliable Datagram service shall decrement its 3 bit retry counter each time the responder returns a NAK-Sequence error for a given request packet. The counter shall be re-loaded whenever the given outstanding request is cleared. If automatic path migration is not supported, and if a NAK-Sequence error is returned once more, then the requester shall declare a locally detected error.

o9-82: If a CA supports automatic path migration, then the following is required. If a NAK-Sequence error is returned after the retry counter has decremented to zero, then the channel adapter shall attempt an automatic

path migration. Following the automatic path migration, the requester shall reload the retry counter and begin the process over again. If the requester still does not succeed in sending the request after several retries, then the requester shall declare a locally detected error.

For other NAK packets, the response of the send queue depends on whether the queue is providing Reliable Datagram or Reliable Connected service.

Reliable Datagram Behavior: Reliable datagrams require the use of an EE Context that maintains the packet sequence numbers and thus ensures reliable delivery of requests. The rules for responding to a NAK ensure that the current PSN at the requester and the expected PSN at the responder remain in sync. Therefore, the connection between the requester's EE Context and responder's EE Context survives. This allows the connection to continue to service other Send/Receive QPs.

Depending on the cause and the operation in question, the EE context may undertake any of the following options after detecting a failed request packet:

a) It may retry the same failed packet from the same QP (note that not all NAKs can be retried, and for those that can be retried there are limits to the number of times a retry is possible), or

b) It may transition the QP and the EE Context to the Error state, completing the failed request message in error, or

c) It may place the current QP in the SQEr state and generate a RESYNC request according to the rules detailed in Section 9.7.8 Reliable Datagram on page 358. In this case, the failed WQE will usually end up being completed in error.

This last strategy is designed to allow the requester side EE Context to continue in service, thus avoiding the need to tear down the EE Context-to-EE Context connection.

If the "same failed packet" is to be retried, the requester is not required to begin its retransmission sequence beginning with the PSN indicated in the responder's NAK; instead, it may begin its retransmission with an earlier request packet. These earlier request packets are treated by the responder as normal duplicate packets causing no ill side effects.

See section 9.9 Error detection and handling on page 396 for a complete description of the errors and the EE Context's subsequent behavior.

C9-138: For an HCA requester using Reliable Connection service, the requester must receive and discard any duplicate acknowledge messages with no ill side effects.

o9-83: If a TCA requester implements Reliable Connection service, or if a CA requester implements Reliable Datagram service, the requester must receive and discard any duplicate acknowledge messages with no ill side effects.

Reliable Connected Behavior: For reliable connections, the requester has only two possible alternatives when it receives a NAK. It may either retry the same request packet, or it may mark the current WQE as completed in error and notify its client. Note that not all NAKs can be retried.

If the requester retries the same request packet, it is not required to begin its retransmission sequence beginning with the PSN indicated in the responder's NAK; instead, it may begin its retransmission with an earlier request packet. These earlier request packets are treated by the responder as normal duplicate packets causing no ill side effects.

9.7.6.1.3 DETECTING LOST ACKNOWLEDGE MESSAGES AND TIMEOUTS

Under some error conditions the requester does not receive an acknowledge message in response to one or more of its requests. This can occur for one of three reasons:

1) The responder generated an acknowledge message that was subsequently lost in the fabric, or,
2) The responder failed for some reason preventing it from generating an acknowledge message, or
3) The original request message was lost in the fabric before it was received by the responder.

All three of these conditions are detected by the requester as a lost acknowledge message.

Often, these errors are corrected automatically due to acknowledge coalescing; the next acknowledge received by the requester serves to implicitly acknowledge all outstanding requests.

However, there are several cases where a lost acknowledge message is not automatically recovered by the coalesced acknowledge rules. For example, a NAK message lost in the fabric will not be resolved via acknowledge coalescing because the responder side rules require that the responder may have no more than one NAK message outstanding at a given time.

C9-139: For an HCA requester using Reliable Connection service, to detect missing responses, every Send queue is required to implement a Transport Timer to time outstanding requests.

o9-84: If a TCA requester implements Reliable Connection service, to detect missing responses, every Send queue is required to implement a Transport Timer to time outstanding requests.

o9-85: If a CA requester implements Reliable Datagram service, to detect missing responses, every EE Context is required to implement a Transport Timer to time outstanding requests.

Because of variabilities in the fabric, scheduling algorithms and architecture of the channel adapters and many other factors, it is not possible, nor desirable, to time outstanding requests with a high degree of precision. Nonetheless, the Transport Timer is an integral element of the ACK/NAK protocol by providing a deterministic means to detect lost requests or responses.

The requester need not separately time each request launched into the fabric, but instead simply begins the timer whenever it is expecting a response. Once started, the timer is restarted each time an acknowledge packet is received as long as there are outstanding expected responses. The timer does not detect the loss of a particular expected acknowledge packet, but rather simply detects the persistent absence of response packets.

The timer measures the lesser of:

- the time since the requester sent a packet with the AckReq bit set in the BTH,
- or the time since the last valid acknowledge packet arrived.

The operation is as follows.

The requester starts the timer running whenever the timer is not currently running AND:

1) The requester sets the AckReq bit in a Send or RDMA WRITE request or,
2) The requester generates an RDMA READ request or,
3) The requester generates an ATOMIC Operation request.

Thereafter, the requester restarts the timer each time it receives a new inbound acknowledge packet as long as there are still outstanding expected responses.

The timer is stopped whenever there are no outstanding expected responses.

An “expected response” is created by the requester by setting the AckReq bit in a request packet or by generating an RDMA READ request or an

ATOMIC Operation request. An outstanding expected response is a response to any request packet which has the AckReq bit set in the BTH, or any RDMA READ request or ATOMIC Operation request, which has not been acknowledged.

Each QP has a single Local ACK Timeout value associated with it which is used to derive the Transport Timer timeout interval $T_{tr.}$

C9-140: For an HCA requester using Reliable Connection service, the Transport Timer timeout interval, T_{tr} shall be defined to be 4.096 uS * $2^{(\text{Local ACK Timeout})}$. Local ACK Timeout shall be a 5 bit value, with zero meaning that the timer is disabled. The minimum acceptable value of Local ACK Timeout, other than zero, shall be defined by the CA vendor. If a non-zero Local ACK Timeout value is loaded in QP context which is less than the minimum supported by the CA, then the CA may use its minimum value.

C9-141: For an HCA requester using Reliable Connection service, a QP shall provide facilities to detect a timeout condition.

The timeout interval should begin after a request has been scheduled. The timeout condition, To, should be detected in no less than the timeout interval, Tr, and no more than four times the timeout interval, 4Tr.

Thus, $T_{tr} <= T_o <= 4T_{tr.}$

Once a timeout for a given request packet is detected, the requester may retry the request.

C9-142: For an HCA requester using Reliable Connection service, to prevent the requester from retrying the request forever, the requester shall maintain a 3 bit retry counter which is used to count the number of times a particular request packet has been retried and timed out. This counter shall be decremented each time the transport timer expires for a given request packet. The counter shall be re-loaded whenever a given outstanding request is cleared.

See Section 9.9.2.1 Requester Side Error Detection - Locally Detected Errors on page 397 for a full description of the retry counter.

C9-143: For an HCA requester using Reliable Connection service, if automatic path migration is not supported, and if the transport timer expires after the retry counter has decremented to zero, then the requester shall declare a locally detected error.

o9-86: If automatic path migration is supported, and If the transport timer expires after the retry counter has decremented to zero, then the channel

adapter shall attempt an automatic path migration. Following the automatic path migration, the requester shall reload the transport timer retry counter and begin the process over again. If the requester still does not succeed in sending the request after several retries, then the requester shall declare a locally detected error.

o9-87: If a TCA requester implements Reliable Connection service, then the five preceding **HCA** compliance statements (that is, timeout rules for outstanding requests) shall be applicable to that TCA.

o9-88: If a CA requester implements Reliable Datagram service, then the five preceding **HCA** compliance statements (that is, timeout rules for outstanding requests) shall be applicable to that CA. In that case, the functionality described applies to the EE Context rather than the Queue Pair.

9.7.6.1.4 DUPLICATE ACKNOWLEDGEMENTS

9.7.1 Packet Sequence Numbers (PSN) on page 282 describes how duplicate requests are generated. These requests may result in duplicate acknowledgments being returned by the responder. The responder may also send unsolicited Acks that appear to be “Ghost Acks” from the point of view of the requestor.

C9-144: For an HCA requester using Reliable Connection service, if the responder is configured to generate end-to-end flow control credits, then the requester must extract end-to-end flow control credits from a duplicate acknowledgment.

o9-89: If a TCA implements Reliable Connection service, and if the responder is configured to generate end-to-end flow control credits, then the requester must extract end-to-end flow control credits from a duplicate acknowledgment.

C9-145: For an HCA requester using Reliable Connection service, duplicate acknowledgments shall be discarded.

o9-90: If a TCA requester implements Reliable Connection service, duplicate acknowledgments shall be discarded.

See Section 9.7.7.2 End-to-End (Message Level) Flow Control on page 347 for a complete description.

9.7.7 RELIABLE CONNECTIONS

A reliable connection is a connection created between a single local QP and a single remote QP and that can guarantee that messages are deliv-

ered at most once, in order and without corruption (in the absence of unrecoverable errors) between the local and remote QPs.

Table 46 Reliable Connected Service Characteristics

Property / Level of Reliability	Support
Corrupt data detected	Yes
Data delivered exactly once (Except for an unrecoverable error - that is reported to the application)	Yes
Data order guaranteed	Yes
Data loss detected	Yes
RDMA Support	Yes - both Read and Write
State of Send/RDMA WRITE when request completed	Completion on remote end node
ATOMIC Support	Optional
Send with Invalidate Operation Type Support	Optional
Multi-packet message support	Yes
Number of messages in flight per QP	2^{23} (maximum)
Number of packets allowed in flight per QP	2^{23} (maximum)
Number of messages enqueued per QP	Implementation limited only
Maximum Message Size	2^{31}Bytes

The desired reliability characteristics are provided by application of packet sequence numbers and the ACK/NAK protocol.

C9-146: For an HCA, each QP configured for Reliable Connection service must conform to the requirements specified in section 9.7 Reliable Service on page 280, the characteristics given in Table 46 Reliable Connected Service Characteristics on page 342, and any additional requirements given in this section.

o9-91: This compliance statement is obsolete and has been removed.

9.7.7.1 GENERATING MSN VALUE

For Reliable Connected service, the Message Sequence Number is a number returned by the responder to the requester indicating the number of messages completed by the responder. The MSN is carried in the three least significant bytes of the AETH. The MSN is provided to the requester as a service to assist it in completing WQEs by informing the requester of the messages that have been completed by the responder.

C9-147: An HCA responder using Reliable Connection service shall return an MSN in the AETH of any response packet which contains an AETH.

o9-92: If a TCA responder implements Reliable Connection service, it shall return an MSN in the AETH of any response packet which contains an AETH.

Logically, the requester associates a sequential Send Sequence Number (SSN) with each WQE posted to the send queue. The SSN bears a one-to-one relationship to the MSN returned by the responder in each response packet. Therefore, when the requester receives a response, it interprets the MSN as representing the SSN of the most recent request completed by the responder to determine which send WQE(s) can be completed.

Note that SSN as described above is a logical concept only which is given to convey the concept of how the MSN is applied; an implementation is not required to implement it as described.

Following initialization, the first WQE posted to the Send queue has an SSN of one assigned to it. The responder initializes its MSN counter to zero. Thereafter, the responder increments its 24-bit MSN value whenever it completes execution of an inbound request message. This is illustrated in Figure below.

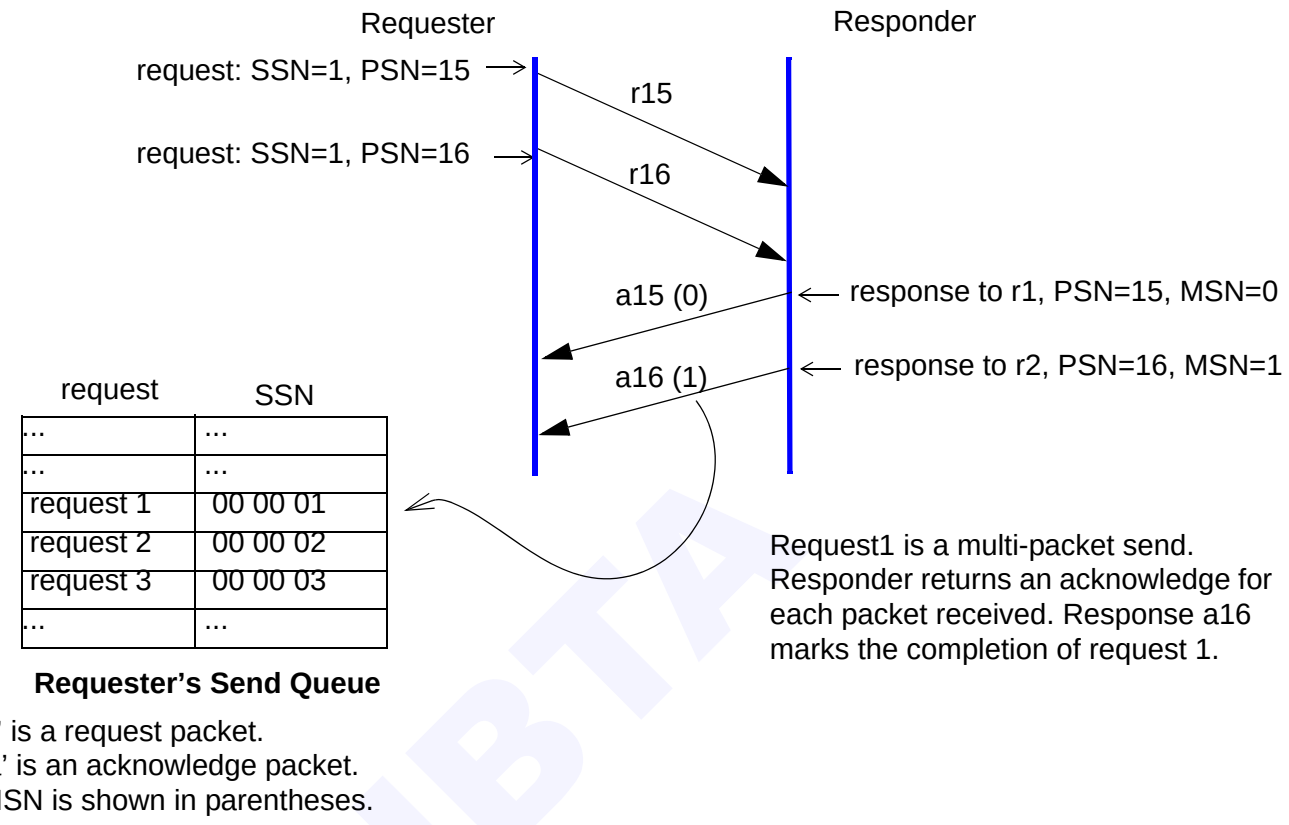


Figure 100 Responder Initializes MSN to Zero

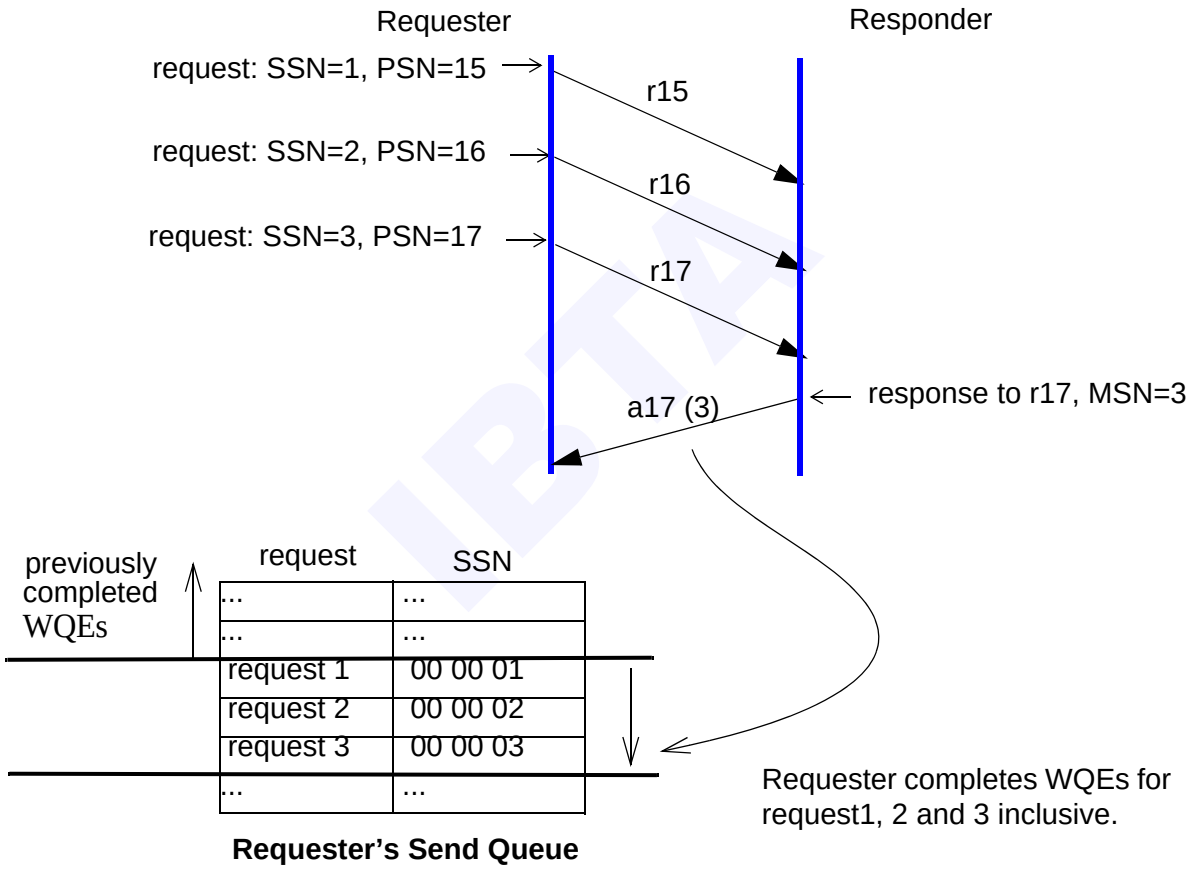
C9-148: An HCA responder using Reliable Connection service shall initialize its MSN value to zero. The responder shall increment its MSN whenever it has successfully completed processing a new, valid request message. The MSN shall not be incremented for duplicate requests. The incremented MSN shall be returned in the last or only packet of an RDMA READ or Atomic response. For RDMA READ requests, the responder may increment its MSN after it has completed validating the request and before it has begun transmitting any of the requested data, and may return the incremented MSN in the AETH of the first response packet. The MSN shall be incremented only once for any given request message.

o9-93: If a TCA responder implements Reliable Connection service, it shall calculate and update MSN as described in the preceding compliance statement.

9.7.7.1.1 REQUESTER BEHAVIOR ON RECEIVING A NEW MSN

As described above, the existence of a new MSN value in a response packet may be used by the requester as a signal to complete certain

WQEs posted to its send queue. Since the responder may choose to coalesce acknowledges, a single response packet may in fact acknowledge several request messages. Thus, when it receives a new MSN, the requester begins evaluating WQEs on its send queue beginning with the oldest outstanding WQE and progressing forward. This is illustrated in the figure below for the case where there are no outstanding RDMA READ requests or ATOMIC Operation requests on the send queue.

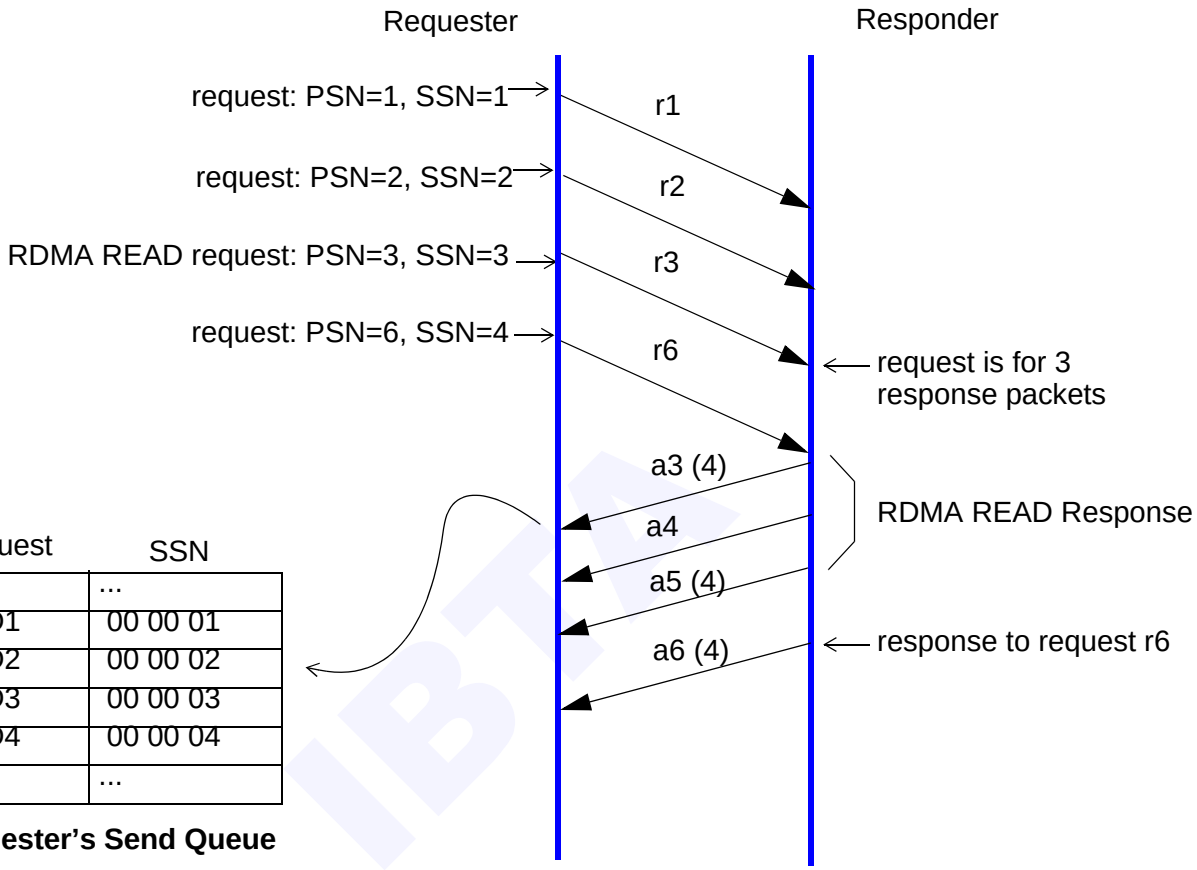


'r' is a request packet.
'a' is an acknowledge packet.
MSN is shown in parentheses.

Figure 101 Requester Behavior - Completing WQEs

For the case where there are outstanding RDMA READ requests or ATOMIC Operation requests, the situation is slightly more complex. In this case, the requester only completes outstanding WQEs up to either the first outstanding RDMA READ request, ATOMIC Operation request, or

WQE whose SSN matches the MSN in the response packet, whichever comes first. This is because both RDMA READ requests and ATOMIC



Since RDMA READs are loosely ordered, it is likely that the responder will "complete" SEND4 before it finishes returning the READ response data (a3, a4, a5). Nonetheless, a3 has an MSN of 4 indicating that it the responder has completed SEND1, SEND2 and SEND4.

However, the requester may only complete SEND1 and SEND2 because of the presence of READ3 in the send queue.

'r' is a request packet
'a' is an acknowledge packet (message)
MSN is shown in parentheses.

\

Figure 102 Limitation on Completing Send Queue WQEs

Operation requests require an explicit response and thus cannot be completed until such an explicit response is received.

C9-149: For an HCA responder using Reliable Connection service, the MSN counter shall be inserted in the AETH regardless of whether the response is a positive acknowledgment, a negative acknowledgment or a duplicate acknowledgment.

o9-94: If a TCA responder implements Reliable Connection service, the MSN counter shall be inserted in the AETH regardless of whether the response is a positive acknowledgment, a negative acknowledgment or a duplicate acknowledgment.

9.7.7.2 END-TO-END (MESSAGE LEVEL) FLOW CONTROL

IBA provides an end-to-end (or message level) flow control capability for reliable connections that can be used by a responder to optimize the use of its receive resources. Essentially, a requester cannot send a request message unless it has appropriate credits to do so.

Encoded credits are transported from the responder to the requester in an acknowledge message in the Syndrome field of the AETH. The credits carried in the AETH are with respect to the MSN field of the same AETH; therefore proper interpretation of the credit field also requires interpretation of the MSN field. See Section 9.7.5.2 AETH Format on page 324 for a full description of the appropriate AETH fields.

Each credit represents the receive resources needed to receive one inbound request message. Specifically, each credit represents one WQE posted to the receive queue. The presence of a receive credit does not, however, necessarily mean that enough physical memory has been allocated. For example, it is still possible, even if sufficient credits are available, to encounter a condition where there is insufficient memory available to receive the entire inbound message.

The shared receive queue concept, on the other hand, allows a set of receive queues to draw from a common pool of receive WQEs - the shared receive queue. Thus, there is no practical way for any of the individual receive queues to accurately gauge the number of WQEs available in the shared receive queue. For this reason, the end-to-end flow control mechanism is disabled for any QP which is associated with a shared receive queue.

1) The end-to-end credit mechanism applies only to Reliable Connected service.
2) End-to-End credits are generated by a responder's receive queue and consumed by a requester's send queue.
3) Requirements on a CA for supporting end-to-end flow control are given in Chapter 17: Channel Adapters on page 1016. HCA receive queues must generate end-to-end credits (except for QPs that are associated with a Shared Receive Queue), but TCA receive queues are not required to do so. If the TCA's receive queue generates End-to-End credits, then the corresponding send queue must receive and respond to those credits.

4) Credits are issued on a per message basis, without regard to the size of the message.
5) End-to-End credits are carried in the AETH as an encoded 5-bit field.
6) The responder may send credits to the requester asynchronously by using an Unsolicited acknowledge packet. An unsolicited acknowledge packet is created by re-sending the most recently sent acknowledge packet.

C9-150: This compliance statement is obsolete and has been replaced by C9-150.2.1:.

C9-150.2.1: For QPs that are not associated with an SRQ, each HCA receive queue shall generate end-to-end flow control credits. If a QP is associated with an SRQ, the HCA receive queue shall not generate end-to-end flow control credits.

o9-95: This compliance statement is obsolete and has been replaced by o9-95.2.1:

o9-95.2.1: Each TCA receive queue may generate end-to-end credits except for QPs that are associated with an SRQ. If a TCA supports SRQ, the TCA must not generate End-to-End Flow Control Credits for QPs associated with an SRQ.

C9-151: If a TCA's given receive queue generates End-to-End credits, then the corresponding send queue shall receive and respond to those credits. This is a requirement on each send queue of a CA.

9.7.7.2.1 TRANSFERRING CREDITS FROM RESPONDER TO REQUESTER

Two mechanisms are defined for transporting credits from the responder's receive queue to the requester's send queue. The credits can be piggybacked onto an existing acknowledge message, or a special unsolicited acknowledge message can be generated by the responder. Piggybacked credits are those credits that are carried in the AETH field of an already scheduled acknowledge packet.

Piggybacked Credits:

Piggybacking of end-to-end credits refers to transferring credits to the requester in the AETH of a normal acknowledge packet. Credits are carried in AETH Syndrome[4:0]. Credits can be piggybacked onto any acknowledge packet when the MSN field in the AETH is also valid.

Unsolicited Acknowledge Packet:

From a PSN perspective, an unsolicited acknowledge message appears to the requester like a duplicate of the most recent positive acknowledge

message. However, it always has an opcode of Acknowledge even if the most recent positive acknowledge was an RDMA READ Response or Atomic Response. Since the ACK/NAK protocol prohibits the responder from sending duplicate negative acknowledge packets (NAKs), an unsolicited acknowledge cannot be created by re-sending a NAK packet.

An unsolicited acknowledge may be sent by the responder at any time. The requester's send queue simply recovers the credit field and the MSN from the most recently received acknowledge packet.

Since an unsolicited acknowledge packet appears to the requester as a duplicate response, it has no effect on the requester other than the transfer of the credits.

C9-152: The MSN field of the unsolicited acknowledge packet must have a valid MSN field.

9.7.7.2.2 NEGOTIATING CONNECTIONS: INITIAL CREDITS

For each connection established, the use (or not) of end-to-end flow control is established separately for each direction. The capabilities of the receive queue determine the flow control characteristics for that half of the connection.

C9-153: If the receive queue signals that it is expecting to generate credits, then the corresponding send queue must observe the end-to-end flow control rules. If, on the other hand, the receive queue signals that it will not generate end-to-end flow control credits, then the corresponding send queue may transmit request messages at will without regard for credits. This is a requirement on each send queue of a CA.

C9-154: If a TCA's receive queue does not generate End-to-End credits, it shall place the value 5b11111 in AETH Syndrome[4:0] signalling that the credit field is invalid.

o9-95.2.2: If a CA supports SRQ and a QP provides reliable connection service and is associated with an SRQ, then the QP must not generate end-to-end credits and shall place the value 5b11111 in AETH Syndrome[4:0] signalling that the credit field is invalid.

C9-155: When the receive queue is in the RESET state, the transport shall set the initial credit count to zero. Once the queue pair has transitioned to the INITIALIZED, RTR, SQD or RTS states, it shall increment its credit count for each receive WQE posted.

Once it is in the RTR, SQD or RTS states, the responder may transfer these credits to the requester by using unsolicited acknowledges.

Normally an unsolicited acknowledge is created by re-sending the most recently sent positive acknowledge packet with an updated credit field. At initialization time however, no acknowledge packets have yet been sent so the normal method for creating an unsolicited acknowledge cannot be used. Therefore, at initialization time, an unsolicited acknowledge is created by subtracting “1” from the initial PSN. Thus, if the PSN is initialized to 0x000000 when the receive queue is in RESET state, then the PSN of the initial unsolicited acknowledge shall be 0xFFFFFF. “Initialization time”, in this context means the interval beginning when the receive queue has transitioned out of the RESET state and has not yet sent an acknowledge packet in either the RTR, SQD or RTS states.

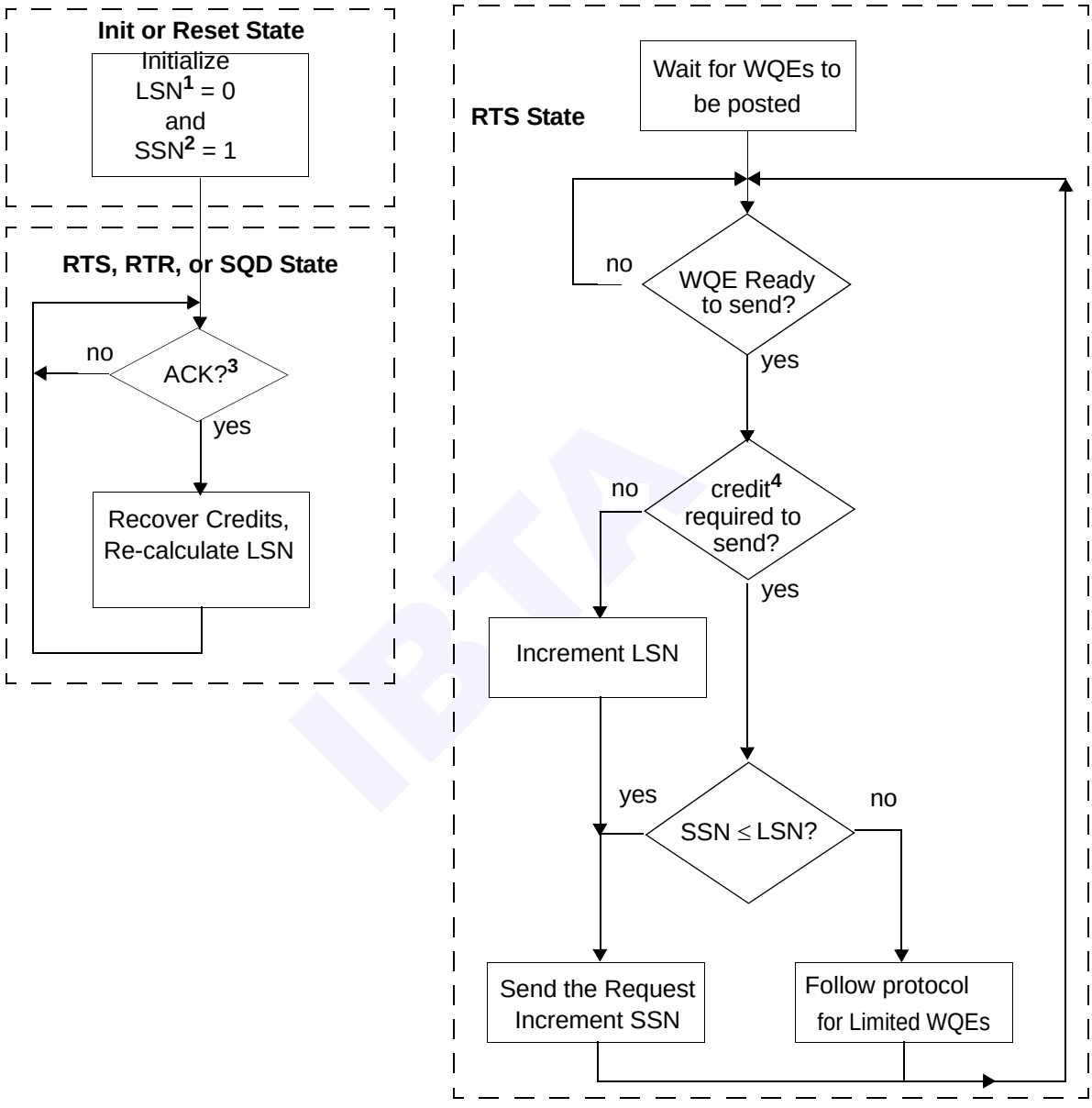
To the send queue which receives this initial unsolicited acknowledge packet, it will appear as a “ghost” acknowledge packet Figure 98 Response Packet PSN Regions on page 333. The requester’s send queue may accept the MSN and credits contained in the unsolicited acknowledge packet but ignore the rest of the packet. This is an exception to the normal rules for ghost responses which require that ghost acknowledge packets be dropped.

The above paragraph notwithstanding, responsibility for recovering initial credits from the responder shall lie with the requester; if the responder provides initial credits by using an unsolicited acknowledge, the requester may accept those as its initial credits in satisfaction of its responsibility to recover initial credits.

C9-156: If the responder does not provide initial credits, the requester shall behave as specified in Section 9.7.7.2.5 Requester Behavior - Limited Send WQEs on page 357

Figures Figure 103 Requester End-to-End Credit Processes on page 351 and Figure 104 Responder End-to-End Credit Initialization Process on page 352 describe this behavior. Note that these figures do not depict all

normal state transitions for the receive and send queues. These are fully specified in the Software Interface chapter.



1) LSN = Limit Sequence Number. This is described below.
2) SSN = Send Sequence Number. This is described below.
3) Any ACK or unsolicited ACK with a valid MSN
4) a credit is required for a “Send” or “RDMA Write with immediate”.

Figure 103 Requester End-to-End Credit Processes

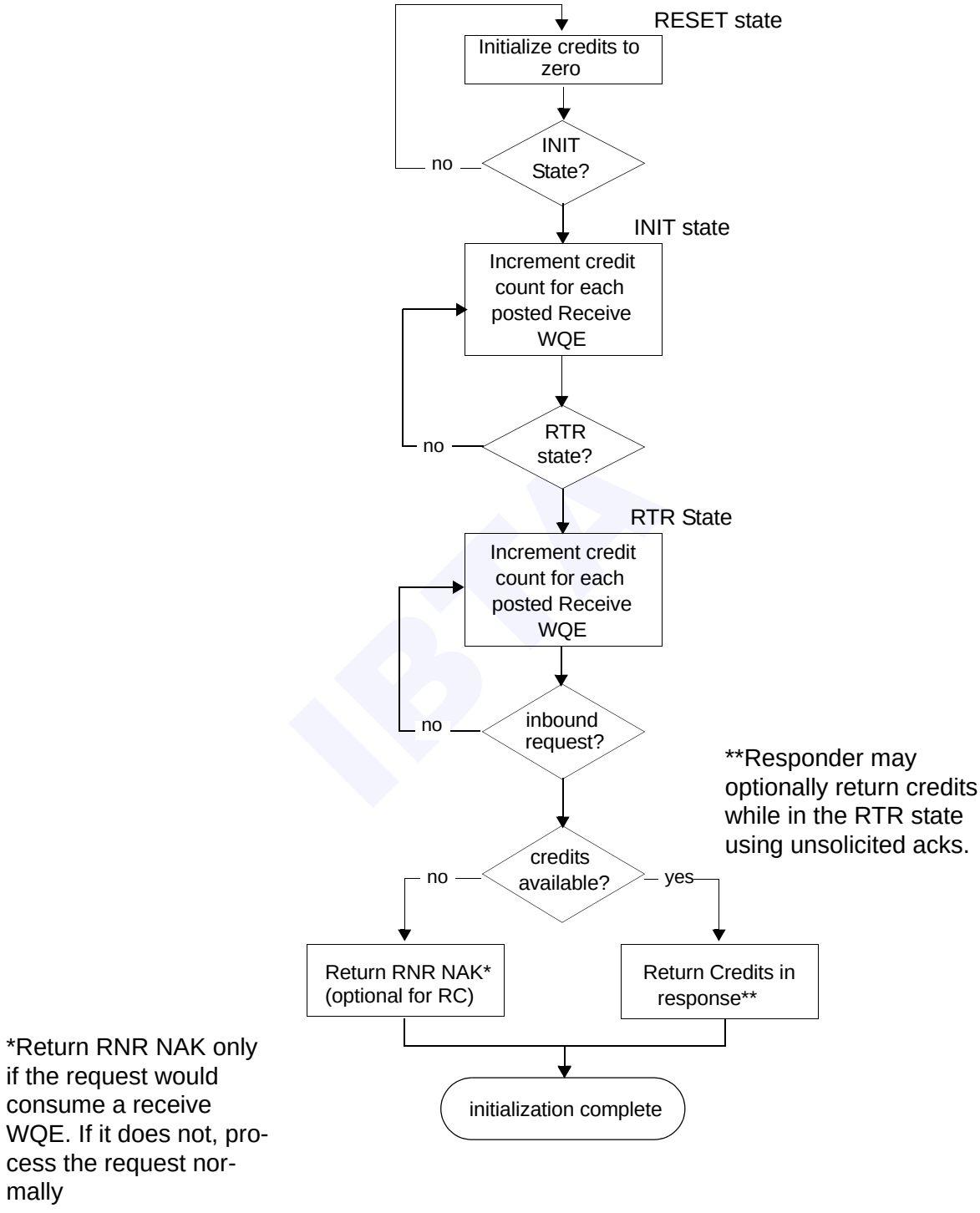


Figure 104 Responder End-to-End Credit Initialization Process

9.7.7.2.3 RESPONDER ALGORITHM FOR CALCULATING CREDITS

C9-157: This compliance statement is obsolete and has been replaced by C9-157.2.1:.

C9-157.2.1: For an HCA using Reliable Connection service on a QP that is not associated with an SRQ, the receive queue shall increment its credit count for each WQE posted to the receive queue. It shall decrement its credit count for each inbound request message received which consumes a WQE. Thus, the responder does not adjust its credit count when it receives an inbound RDMA READ request, an RDMA WRITE request without Immediate data or an ATOMIC Operation request. If the receive queue is associated with an SRQ, the responder does not adjust its credit count regardless of the WQEs that may be posted to the receive queue.

o9-96: If a TCA implements Reliable Connection service, and if the receive queue generates end-to-end flow control credits, it shall increment its credit count for each WQE posted to the receive queue. It shall decrement its credit count for each inbound request message received which consumes a WQE. Thus, the responder does not adjust its credit count when it receives an RDMA READ request, an RDMA WRITE request without Immediate data or an ATOMIC Operation request.

C9-158: This compliance statement is obsolete and has been replaced by C9-158.2.1:.

C9-158.2.1: For an HCA using Reliable Connection service on a QP that is not associated with an SRQ, for each acknowledge message generated, either a normal acknowledge message or an unsolicited acknowledge message, the receive queue shall insert its current encoded credit count as shown in Table 47 End-to-End Flow Control Credit Encoding on page 354, in AETH Syndrome[4:0]. For example, if the receive queue has five credits available, it shall insert the 5 bit value b00100 in the AETH. It also includes its current MSN value. If the QP is associated with an SRQ, the receive queue shall insert the 5 bit value 5b11111 in the AETH

o9-97: If a TCA implements Reliable Connection service, and if the receive queue generates end-to-end flow control credits, for each acknowledge message generated, either a normal acknowledge message or an unsolicited acknowledge message, it shall insert its current encoded credit count as shown in Table 47 End-to-End Flow Control Credit Encoding on page 354, in AETH Syndrome[4:0]. For example, if the receive queue has five credits available, it shall insert the 5 bit value b00100 in the AETH. It also includes its current MSN value. If a TCA does not generate end-to-end flow control, it shall insert the value as shown in Table 47 End-to-End Flow Control Credit Encoding on page 354 to indicate that the credit count field is invalid.

9.7.7.2.4 REQUESTER BEHAVIOR

The presence or absence of credits limits the sender's ability to transmit requests which will consume a receive WQE (SEND requests or RDMA WRITE requests with immediate data).

C9-159: The send queue's behavior when it has no credits available to it shall be as specified in Section 9.7.7.2.5 Requester Behavior - Limited Send WQEs on page 357.

The requester may always send a request which does not consume a receive WQE (RDMA WRITE request without immediate data, RDMA READ request, or ATOMIC Operation request) without regard to credits.

C9-160: The requester shall not violate the normal transaction ordering rules as stated throughout this specification, particularly in Section 9.5 Transaction Ordering on page 268.

In particular, the requester may not search the send queue looking for requests which don't consume a receive WQE and transmit those requests out of order, nor may the requester violate the rules governing fenced WQEs.

The available credits are encoded and carried in AETH Syndrome[4:0]; the MSN is carried in the least significant 3 bytes of the AETH. Table 47 below shows, for each valid encoded credit, the actual number of credits.

Table 47 End-to-End Flow Control Credit Encoding

Credit	Valued added to MSN to get LSN	Credit	Valued added to MSN to get LSN
00000	0	10000	256
00001	1	10001	384
00010	2	10010	512
00011	3	10011	768
00100	4	10100	1024
00101	6	10101	1536

Table 47 End-to-End Flow Control Credit Encoding

Credit	Valued added to MSN to get LSN	Credit	Valued added to MSN to get LSN
00110	8	10110	2048
00111	12	10111	3072
01000	16	11000	4096
01001	24	11001	6144
01010	32	11010	8192
01011	48	11011	12288
01100	64	11100	16384
01101	96	11101	24576
01110	128	11110	32768
01111	192	11111	invalid

Logically, the requester associates a sequential Send Sequence Number (SSN) with each WQE posted to the send queue. The SSN bears a one-to-one relationship to the MSN returned by the responder in each response packet. Thus, the requester interprets the MSN as representing the SSN of the most recent request completed by the responder.

C9-161: The encoded credit count returned by the responder in the AETH shall specify the number of receive WQEs posted to the responder's receive queue relative to the MSN unless the responder does not generate end-to-end flow control credits, in which case the "invalid"code is carried in the AETH.

Since the MSN is directly related to the requester's SSN, the credit count is a simple offset into the send queue from the SSN of the most recent request completed by the responder. Logically, the sum of the MSN plus the credit count is the requester's Limit Sequence Number (LSN). The requester may freely transmit any request whose SSN is less than or equal to the computed LSN.

If the responder returns the "invalid" code in the AETH instead of the credit count, then the requester may freely transmit requests at will. The "invalid" code is used by the responder to indicate that the AETH credit count field is not valid because the responder does not generate end-to-end credits. Even a responder which does generate end-to-end credits may choose to send the "invalid" code in the AETH. However, once a requester has received an "invalid" code in the AETH from the responder, the requester may choose to ignore the AETH credit count field for all future transactions on that connection. Thus, if a responder resumes returning valid credits after having signalled "invalid", the results may be unpredictable.

Any request whose SSN is greater than the current computed LSN is said to be limited. The send queue's behavior when it encounters a limited request is as specified in Section 9.7.7.2.5 Requester Behavior - Limited Send WQEs on page 357.

Figure 105 Relating AETH values to the Send Queue on page 356 illustrates the relationship between the values returned by the responder in the AETH and the requester's send queue.

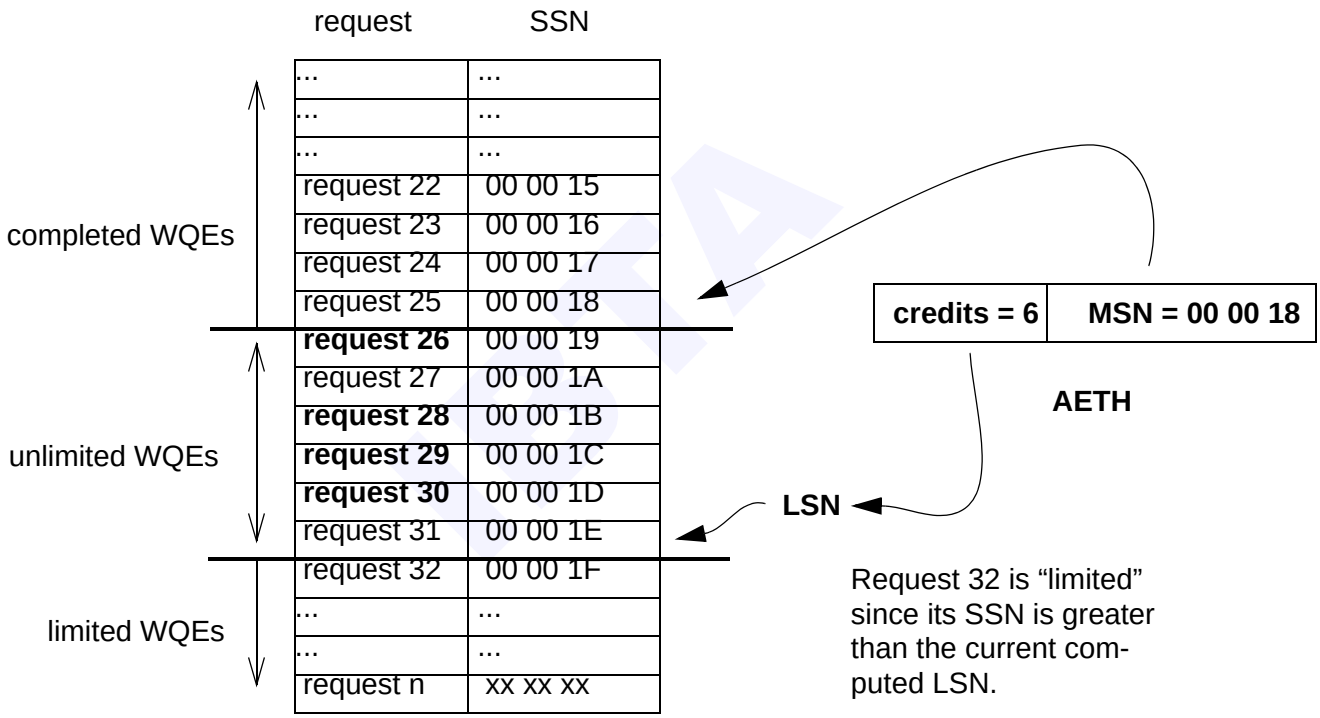


Figure 105 Relating AETH values to the Send Queue

The requester calculates a new LSN each time it receives an acknowledge packet containing valid credits. The requester also dynamically adjusts the LSN by adding one to it for every request it wishes to send that does not consume a receive WQE (RDMA READ requests, RDMA WRITE requests without immediate data, or ATOMIC Operation requests). This adjustment is the mechanism which allows the requester to send requests that do not consume a receive WQE.

Any given implementation is not required to implement the LSN and SSN mechanisms described above, but must conform semantically to the behavior described.

9.7.7.2.5 REQUESTER BEHAVIOR - LIMITED SEND WQES

C9-162: When the requester encounters a WQE on its send queue for which it has no available credits, that WQE is said to be limited. The send queue's behavior when it encounters a limited WQE shall be as follows:

- If the limited request WQE is an RDMA READ request, an RDMA WRITE request without immediate data, or an ATOMIC Operation request, it may be sent normally without regard to the availability of credits. The normal rules for ordering of requests still hold (i.e., the send queue may not search through the list of posted WQEs in an attempt to find unlimited WQEs to be sent out of order). After sending such a request, the requester increments its computed LSN value[11], since the sent request does not consume a receive WQE and thus does not consume a credit.
- If the limited request WQE is a SEND request, the send queue shall transmit no more than a single packet of the request message before it must stop transmission and wait for an acknowledge packet. To ensure that the responder will generate a response, the requester shall set the AckReq bit in that single packet.
- If the limited request WQE is an RDMA WRITE request with immediate data, the requester may transmit the entire request message before it must stop transmission and wait for an acknowledge packet. This is permitted because it is the single packet containing immediate data of the request that actually consumes the receive WQE. To ensure that the responder will generate a response, the requester shall set the AckReq bit in the last packet of the request message.

C9-163: For an HCA using Reliable Connection service, if the limited WQE is a SEND request, the send queue shall transmit no more than a single packet of the request message. Within this single packet, the Acknowledge Request (AckReq) bit of the BTH shall be set. The requester shall then stop transmission and wait for an acknowledge packet.

o9-98: If a TCA implements Reliable Connection service, and if the limited WQE is a SEND request, the send queue shall transmit no more than a single packet of the request message. Within this single packet, the Acknowledge Request (AckReq) bit of the BTH shall be set. The requester shall then stop transmission and wait for an acknowledge packet.

11. An interesting situation can occur that artificially limits the sender LSN with certain message patterns; if the sender does Send, RDMA, RDMA, RDMA with two credits from the receiver, it will increment the LSN by three. If after that, the response arrives with MSN+1 credit, the LSN will then be set back by two, putting the requestor into limit until the Ack from the RDMA's arrive.

o9-99: In an HCA using Reliable Connection service, or if a TCA implements Reliable Connection service, and if the limited request WQE is a RDMA WRITE request, the requester may transmit the entire request message before it must stop transmission and wait for an acknowledge packet. To ensure that the responder will generate a response, the requester shall set the Acknowledge Request (AckReq) bit in the last packet of the RDMA WRITE request.

C9-164: Since the responder's receive queue may generate an unsolicited acknowledge message at any time, the requester shall be prepared to receive an unsolicited acknowledge message from the responder at any time, provided that the receive queue has signalled that it will generate end-to-end flow control credits.

An unsolicited acknowledge is used solely for the purpose of transferring credits from the responder to the requester. On receiving an unsolicited acknowledge, the requester recalculates its LSN as specified above and responds accordingly. A lack of credits does not impact a requester's ability to re-transmit previously transmitted requests as part of its recovery from lost packets. End-to-end credits only limit the transmission of new request messages. For example, if the requester detects a timeout condition after having sent a single packet of a limited SEND request, it decrements its timeout retry counter as usual and retransmits the request.

9.7.8 RELIABLE DATAGRAM

Reliable Datagram provides reliable communication, i.e. the same level of reliability and error recovery as for Reliable Connection, using a one-to-many paradigm. A requestor's send queue may send sequential messages to different responders, at different QPs on the same or different nodes. A responder QP may receive messages from multiple requesters on the same or different endnodes. As with the Unreliable Datagram transport service, the source endnode and source QP are provided to the responder.

The motivation for using Reliable Datagram is to economize the QP name space for applications that engage in "all to all" communication. Consider N processor nodes, each with M processes. If all M processes wish to communicate with all the processes on all the nodes, a Reliable Connection service requires $M^2*(N-1)$ QPs on each node. By comparison, the Reliable Datagram service only requires M QPs + N "end-to-end" (EE) connections on each node for exactly the same communications.

Reliability is implemented using at least one "QP-like" context for each remote endnode - this is referred to as the End-to-End Context (EE Context or EEC). This context provides the information needed to locate the remote node, to serialize and exchange acknowledgments, and maintain reliability.

The Service still uses the QPs to provide the queues, WQE pointers, protection checking parameters etc. Together, a QP and an EEC contain the information needed to reliably move messages to a destination. But many QPs may use a single EEC for sending or receiving, and a QP may communicate through several different EECs, one chosen with each message.

When an application determines the target that it is to communicate with, it must first establish (or use an already established) an EE Context.

As with Reliable Connection, the local QPs to use for this service are established in the RD service mode by the application prior to use. Remote QPs are chosen in an application dependent manner.

Once the EE Context is created, the client may send a message to the responder QP via this EE Context. The client must specify the local EE context number, the responder QP, the Q_Key, and any additional message parameters. The implementation then “multiplexes” the messages from each source QP to the appropriate EEC, and sends the message. When the message arrives at the destination, the implementation uses the EE Context to validate the packet and “de-multiplexes” the message to the appropriate QP.

The Reliable Datagram service uses the methods (PSNs, ACK/NAK protocol etc.) as described previously in 9.7 Reliable Service on page 280.

9.7.8.1 RELIABLE DATAGRAM CHARACTERISTICS

Table 48 Reliable Datagram QP characteristics

Property / Level of Reliability	Support
Corrupt data detected	Yes
Data delivered exactly once (Except for an unrecoverable error; that is reported to the application)	Yes
Data order guaranteed to same destination QP from the same source QP	Yes
Data order guaranteed to different destinations from the same QP	Yes
Scalability (number of messages) on the service	Limited to number of EE Contexts in use between endnodes
Data loss detected	Yes
RDMA READ Support	Yes
RDMA WRITE Support	Yes
State of SEND/RDMA WRITE when request completed	Completion on remote end node

Table 48 Reliable Datagram QP characteristics (Continued)

State of in-flight SEND/RDMA WRITE when unrecoverable error occurs	First one unknown, others not delivered
ATOMIC Support	Optional
Multi-packet message support	Yes
Multiple EE Context allowed between end-nodes (to provide traffic segregation for QOS)	Yes
Single SL / QoS assigned to EE Context	Yes
Number of messages in-flight per EEC	1
Number of messages in flight per QP	1
Number of messages enqueued per EEC / QP	Implementation limited only
Number of packets allowed in flight (architectural)	2^{23}
RD QP shall only communicate with RD QPs	Yes
Partition Key verification	On a per EEC basis
Protection verification (e.g. Q_Key, R_Key, etc.) and Addressing	On a per QP basis
Max Size of messages	2^{31}
Destination QP, Q_Key, and address supplied	On a per send WR basis
Source QP and address supplied	On a per receiver completion basis

o9-100: CA's claiming to support RD mode shall provide QP's capable of supporting RD. When operating in RD mode, these QPs allow sending sequential RD messages to different responders, at different destination QPs on the same or different nodes. When operating in RD mode, these QPs shall be capable of receiving RD messages from multiple requesters on the same or different endnodes.

o9-101: CA's claiming to support RD mode shall provide EEC's that allow the "multiplexing" of multi packet RD message traffic to and from multiple QPs while maintaining reliability (messages are delivered from a requester to a responder at most once, in order and without corruption, or the upper layer is notified.)

o9-102: CA's claiming to support RD mode shall ensure that an RD message has been completed at the sender (fully acknowledged or completed in error) before sending another message on the same EEC.

o9-103: CA's claiming to support RD mode shall ensure that an RD message has been completed at the sender (fully acknowledged or completed in error) before sending another message on the same QP.

o9-104: CA's claiming to support RD mode shall meet the requirements specified in 9.2.1 Operation Code (OpCode) on page 234 for coding of the RD OpCodes, 9.3.1 Reliable Datagram Extended Transport Header (RDETH) - 4 Bytes on page 240 for creation of that header, and 9.6 Packet

Transport Header Validation on page 269 and 9.7 Reliable Service on page 280 through 9.7.6 for reliable transports for processing RD messages.

o9-105: CA's claiming to support RD mode shall meet the requirements specified in 9.9 Error detection and handling on page 396 while processing RD messages.

o9-106: CA's claiming to support RD mode shall provide support for setting up connections between EECs as defined in Chapter 12: Communication Management on page 650, using the Management facilities as defined in Chapter 13: Management Model on page 709

o9-107: CA's claiming to support RD mode shall ensure that RD message errors or events that are not associated with the underlying EE Context (for example Q_Key or R_Key violations or RNR-NAK) shall not cause that EE Context to shut down or prevent the EE Context from processing other RD messages destined to other QPs.

o9-108: HCA's claiming to support RD mode shall provide support for Send, RDMA WRITE, RDMA READ, and ATOMICS in RD mode to the extent defined and reported in 11.2 Transport Resource Management on page 550.

o9-109: HCA's claiming to support RD mode shall provide support for EEC management as defined in 10.2.7 End-to-End Contexts on page 441 and 11.2.7 EE Context on page 584.

o9-110: HCA's claiming to support RD mode shall provide support for RDD domains as defined in 10.2.8 Reliable Datagram Domains on page 443, 11.2.1.7 Allocate Reliable Datagram Domain on page 558, and 11.2.1.8 Deallocate Reliable Datagram Domain on page 559.

Implementation note: For many implementations, an EEC will actually be a special "mode" of a general QP or EE context. For these implementations, the context number specified as a destination EEC must be set up in Reliable Datagram 'EEC' mode. Reliable Datagram packets arriving at a context (identified by the EE Context field in the header) that is not set up to "EE Context" mode, shall be silently dropped.

The responder QP context must be set to support Reliable Datagram transport service. If a Reliable Datagram packet arrives at a QP context that is not configured for RD operation, the responder shall respond with a "NAK Invalid RD Request".

An important distinction for this service is that errors that are not associated with the underlying EE Context do not result in shutting that EE Context down. Examples of these would be Q_Key or R_Key violations.

Similarly, the Receiver Not Ready (RNR NAK), caused by resources associated with the receiver's QP does not prevent the EE Context from processing other messages destined to other QPs.

Errors that are associated with the EE Context (retry limit exceeded, etc.), detected during a message transmission or reception, shall be reported in the WR completion.

Errors associated with the requestor or responder QP shall be reported in the WR completion with the usual error semantics. See 9.9 Error detection and handling on page 396 for a more complete discussion on errors.

Since an end-to-end credit mechanism is not practical in a "connectionless" type of service, responders shall send a NAK Receiver Not Ready response if a requester's SEND arrives while the responder's Receive Queue is empty. See 9.7.5.2.8 RNR NAK on page 328 for additional details.

To preserve the ordering rules required of this service, and to keep the design complexity down, messages on this service are sent one at a time from the source QP with the requirement that each message acknowledgment be received at the requesting QP before the next message can be started.

9.7.8.2 EXAMPLE RD OPERATIONS

The following is not normative material, but is included to clarify this topic. These examples are based on an HCA implementation; other implementations are possible. TCAs, for instance may not utilize virtual memory and may modify other details of this example.

This implementation example maintains a standard send queue for WRs.

It also maintains a linked list of Send QPs, anchored at each EEC. This list contains QPs, each of which has a WR at the head of its Send Queue that is destined for the EEC.

In order to manage the orderly transmission of packets and messages, the implementation uses a scheduler. This scheduler maintains a list of those EEC that have packets to send. As each EEC gets to the head of the scheduler list, one or more packets are sent (depending on QoS and other factors not important here).

On the responder side, the implementation example maintains a standard set of Receive Queues.

The implementation also maintains space in each EEC to copy those parameters needed to process a single incoming message. These parame-

Two views of “connectionless”, Reliable Datagram service. The figure to the right shows a software view of Reliable Datagram communication among 4 processes on 3 processors. In this example, there is no communication among process E and processes C and D, otherwise, Process A can send to and receive from all the other processes.

The lower parts of the figure shows the multiple EE Contexts used by the CA to synthesize the Reliable Datagram service. Each context shows some of the state it uses to “connect” to the others.

The SendQ 4 state shows the destinations of three messages; the ReceiveQ states show the Queues after successful transmission of those messages.

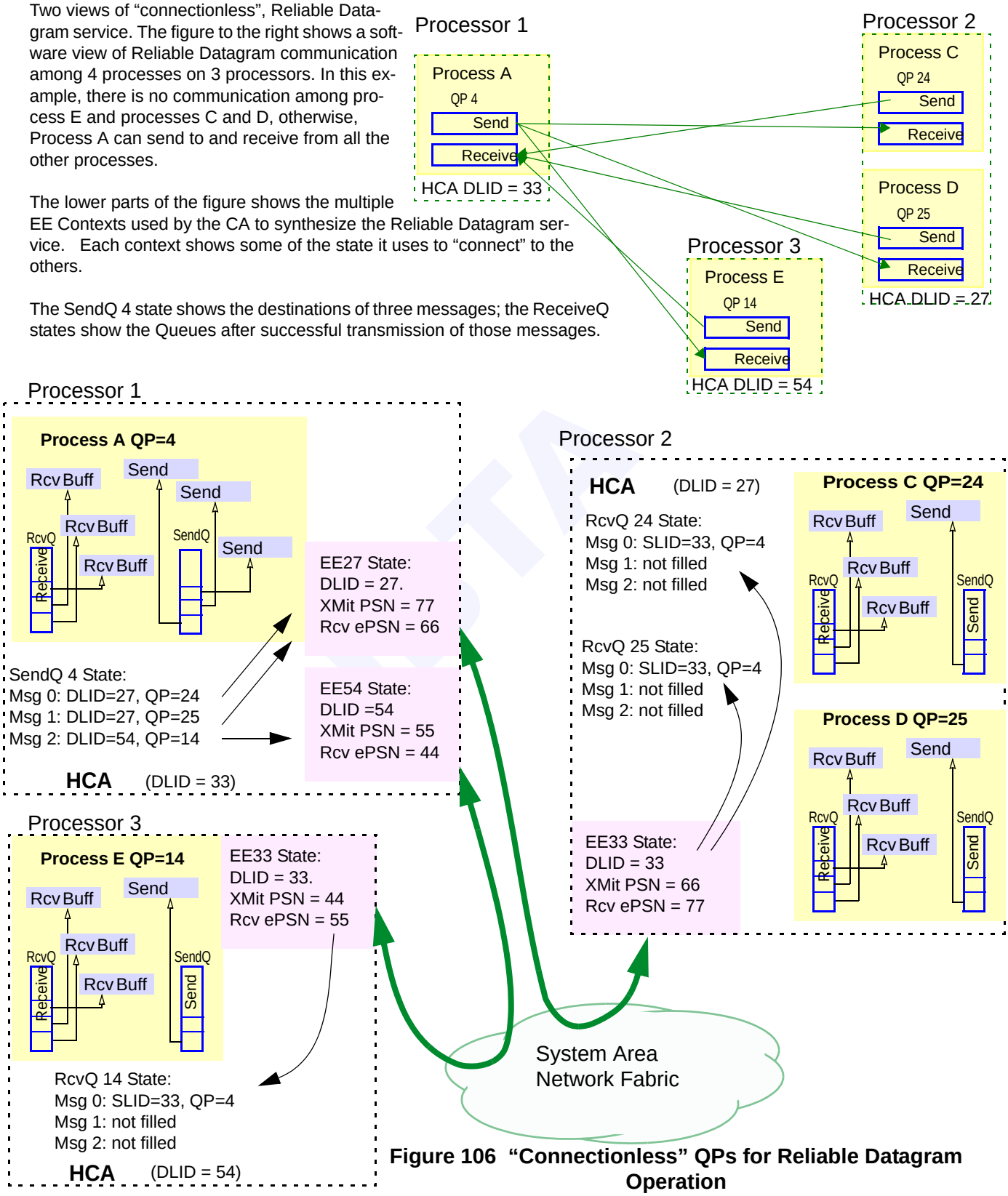


Figure 106 “Connectionless” QPs for Reliable Datagram Operation

ters are copied both from the QP (PD, CQ, Q_Key etc.) and the receive WQE (data segment L_Key, Virtual address, size etc.). The EEC then has enough information to process the entire message to completion with no further reference to the WQE or QP, even if the message contains many packets.

9.7.8.2.1 EXAMPLE OUTBOUND REQUEST

1) The client of the Reliable Datagram posts a send message (described by WQE) to the send queue of its QP. This consists of:
   - the list of data segments (virtual address, L_Key and length) that describes the send message
   - the local EE Context number
   - the destination QP number
   - the destination Q_Key
2) When the WQE reaches the head of the Send Queue (found by pointer from the QP context), the EE Context is located from the WQE and the QP is "linked" to the EECs "QP list" for processing (EEC contains enqueue and dequeue pointers, each QP contains link pointer to next QP to run. Take QP at enqueue pointer, update its "next" link to point to the new QP, adjust enqueue pointer to the newly linked QP).

   If the EEC is not currently sending messages, the EEC is also placed into the scheduler.
3) When the EEC is scheduled to send a message, the HCA locates the WQE parameters by accessing the QP at the head of the EEC's "QP list" (Take QP found at the Dequeue pointer) and using the QP's work queue pointers.
4) HW uses the memory protection parameters of the enqueuing process (stored with the QP Context) and the virtual address etc. from the WQE. This allows the Send Queue HW to directly access the virtual address space of each process that posts send message buffers.
5) The HCA hardware reads the data buffer, builds the transport header (including the "Packet Sequence" number associated with the EE Context) and puts the packet onto the wire.
6) This process is repeated from step 3 until the entire message is sent. The "EE Context" is serviced according to the same scheduling algorithm used for Reliable Connection QPs.
7) When a message is completely sent, the CA waits until all Acknowledgments are in for the message.

8) Since the EEC must wait for a message ACK before continuing (only a single message outstanding at once), the EEC is scheduled with an appropriate timeout and the EE Context is updated.

9) When the last ACK has arrived and the WQE completed, the HCA determines if there are additional WQEs posted to the current QP (the one at the head of the EEC's QP list). If so, the next WQE is examined to locate the EEC for the QP's next message (this may be to a different EEC than the current). The CA then dequeues the QP from the current EEC's QP list, and enqueues it on the tail of the next message's EEC QP list. This is similar to step 2 above.

10) The EEC's QP list is examined to determine if any QP has work for this EEC.

11) The process repeats from step 3 until no more messages are available to send. At this point, the EEC is removed from the scheduler and set into an "inactive" state.

9.7.8.2.2 EXAMPLE INBOUND REQUEST

The inbound request needs to access the QP state associated with the responder's Receive Queue, the receive WQE, and the EE Context that maintains information about the source. Both QP and EEC are available in the header for this purpose.

The following lists the steps taken by the HCA to process an incoming request packet:

First or Only Packets

1) The incoming request packet arrives and is found to be un-corrupted and the first or only packet of the message.

2) The packet header specifies the destination QP number. This is the QP associated with the client of the Reliable Datagram service. This QP points to the receive queue, and a WQE, but does not have any sequence number information. The packet header also includes the "EE Context number" that is used to access the EE Context. The sequence number information is stored with the EE Context connected to the requesting host.

3) The incoming request's sequence number is compared against the state of the EE Context connected to the requesting node.

4) If the sequence number and other packet contents are correct, the destination QP's memory protection and WQE entry information are temporarily copied to the EE Context. This implementation is useful because other EECs may be targeting the same QP and other messages will end up in progress to the same QP. By copying the WQE and memory related information to the EEC, the QP is free to point to subsequent WQEs for additional messages. This is also the reason that receive WQEs may complete out of order.

5) The memory protection checks are done, and if the receive buffer is valid, the incoming request is written to memory (or in the case of a RDMA READ, stored for later processing).

6) The CA puts the EEC on the scheduler to send an ACK response.

7) If the packet was an “only”, then the CA completes the message using the EEC’s copy of WQE and QP values, with no additional references to the QP or WQE.

Middle or Last packets

1) For subsequent packets from the same message, only the EE Context is accessed based on the header EEC number. This allows other EECs to utilize other WQEs from the QP Receive Queue independently.

2) If the sequence number and other header checks are correct, the memory protection checks are done, and if the receive buffer is valid, the incoming request data is written to memory.

3) The CA puts the EEC on the scheduler to send an ACK response.

4) If the packet was a “last”, then the CA completes the message using the EEC’s copy of WQE and QP values, with no additional references to the QP or WQE.

9.7.8.2.3 EXAMPLE OUTBOUND ACKNOWLEDGE

When the EEC gets to the head of the scheduler queue, the CA notes that an ACK must be sent, and sends it. If multiple packets have arrived before the EEC gets to the head of the scheduler, this creates a coalesced ACK. The EE Context’s last valid receive sequence number is sent in the ACK packet per the ACK/NAK rules.

If the operation was an RDMA read, then multiple response packets may be required. In this case, the EEC is placed back on the scheduler after each packet until the PSN of the responses reaches the expected PSN.

9.7.8.2.4 EXAMPLE INBOUND ACKNOWLEDGE

A returning ACK response indicates a request packet was successfully completed. When the ACK arrives, the EE Context is examined and the returned PSN is checked. If this is the expected (next sequential) ACK, the expected PSN is updated. If this is the last ACK of a message and all previous packets were acknowledged, then the message can be completed using the EEC’s copy of the QP and WQE information. If the ACK is not sequential, then the usual coalesced ACK rules apply. Since only a single message is outstanding at one time, only a single message is ever acknowledged at one time.

For RDMA READs, the CA uses the EEC’s copied QP protection information and WQE data segment information to store the data.

9.7.8.3 RELIABLE DATAGRAM OPERATIONS

The processing is very much the same as defined for Reliable connection service. The significant difference is for the treatment of repeated packets at the responder, and the rules for repeating a request at the requester. The differences are highlighted in *italics*.

9.7.8.3.1 SEND AND RDMA WRITE WITH IMMEDIATE DATA PROCESSING

SENDs and RDMA WRITEs with Immediate data are handled in the same way as for Reliable Connection service, *except that end-to-end credits are not returned to the sending QP.*

o9-111: CA's claiming support for Reliable datagram service shall use the NAK-RNR protocol to indicate an over-run of the Receive Queue for RD messages.

9.7.8.3.2 RDMA READ PROCESSING

RDMA READs are handled in the same way as for Reliable Connection service. Incoming requests are stored at the responder's "hidden resources", attached to the EE Context, and memory protection information is accessed or copied from the QP Contexts. *Unlike Reliable Connection service, the number of RDMA READ request messages outstanding from a single QP or EEC shall be limited to one*.

9.7.8.3.3 ATOMICS PROCESSING

Atomics are handled in the same way as for Reliable Connection service. Incoming requests are stored at the responder's "hidden resources", attached to the EE Context, and memory protection information is accessed or copied from the QP Contexts. *Unlike Reliable Connection service, the number of ATOMIC requests outstanding from a single QP or EEC shall be limited to one*.

9.7.8.4 ORDERING RULES

Receive Queues are FIFO queues. Once enqueued, WQEs shall begin processing in FIFO order, but *may be completed out of order*. The messages from any single source QP shall always be in order.

o9-112: CA's claiming to support RD mode shall provide upper layer support for out of order receive queue completion for RD messages.

Send queues are FIFO queues. Once enqueued, WQEs shall be processed for sending in the order they were enqueued.

o9-113: CA's claiming to support RD mode shall ensure that WQEs on the Send Queue in RD mode are completed in order whether they are targeting different destination QPs on the same or a different endnode or the same destination QP. The completions for WQEs shall always be returned to the transport consumer in FIFO order.

This does not mean that the implementation must place the data portions of the messages in memory in any particular order. As a result, the arrival order is not guaranteed until the message is marked complete on at least one side. An application shall expect that memory buffers are undefined until the message is completed.

Note that items queued on different QP's Send Queues on the same HCA for the same destination endnode or even the same destination QP are not ordered with respect to each other. For example, if WQE 'A' destined for destination'X' and QP "75" is posted to QP 1, and WQE 'B' destined for destination'X' and QP "75" is later posted to QP 2 of the same CA, there is no guarantee that 'A' will arrive before 'B' at the destination.

o9-114: For CA's claiming to support RD mode, upper layers must tolerate lack of ordering among RD messages from different send QPs. That is, items queued on different QP's Send Queues on the same HCA for the same destination endnode or even the same destination QP are not ordered with respect to each other.

9.7.8.5 HANDLING QP ERRORS - RESYNC

Since RD service allows multiple QPs to share a single EEC, it is desirable that a QP with an error localized to the QP, not effect the remainder of the QPs sharing the same EEC. To support this, RD service allows the EEC to "Abandon" or "Suspend" operations on a QP under certain error conditions.

A message is "Abandoned" if it is completed in error at the source QP, and the QPn is transitioned to the error state.

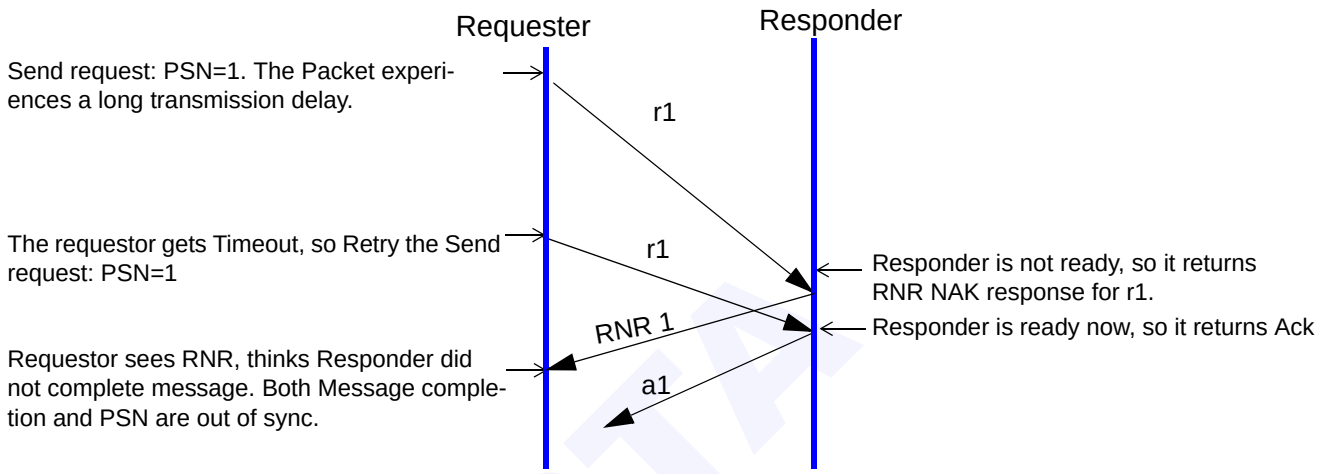
A message is "Suspended" if it is not completed at the source QP, but another message is started on the same EEC. When later resumed, the suspended message must be restarted from the beginning if it is a Send, or RDMA Write with immediate operation. If it is an RDMA Read or RDMA Write w/o Immediate, it can (implementation choice) be restarted where it left off, but must appear to be a new message to the responder.

"Suspend" and Restart only apply to RNR NAK conditions, where the responder is temporarily unable to perform the request associated with a particular QP, and the requestor can improve performance by sending messages from other QPs on the same EEC, while waiting for the RNR NAK timeout.

A requestor is not allowed to Suspend an Atomic operation.

For convenience, we will sometimes say a message is "aborted" when when either "abandoned" or "suspended" is meant.

The concept of “Abandon” or “Suspend” does require the RD service to deal with a class of errors that can occur when packets associated with a message are delayed, repeated, or both. To deal with this problem, a RE-SYNC operation is required whenever a message is “Abandoned” or “Suspended”. An example of this problem is shown in Figure 107 below where an RNR’d request is actually executed by the responder.



’r’ is a request packet
’a’ is an acknowledge packet (message)

Figure 107 Loss of synchronization of the EEC on Suspend

If the requestor simply started a new message at the original PSN, data corruption could occur. In addition, the requestor, when it restarted the RNR NAK’d message later, the responder would get two copies.

To deal with this problem, a RESYNC is used following any error that affects a QP, but leaves the EEC able to continue operation. The RESYNC serves several purposes, it gets the PSNs on both ends of the EEC synchronized, it lets the responder know that the previous message, if incomplete, is to be abandoned, and it allows the requestor to determine with certainty the status of the current message at the responder. The following flow chart shows the resynchronization process at the requestor.

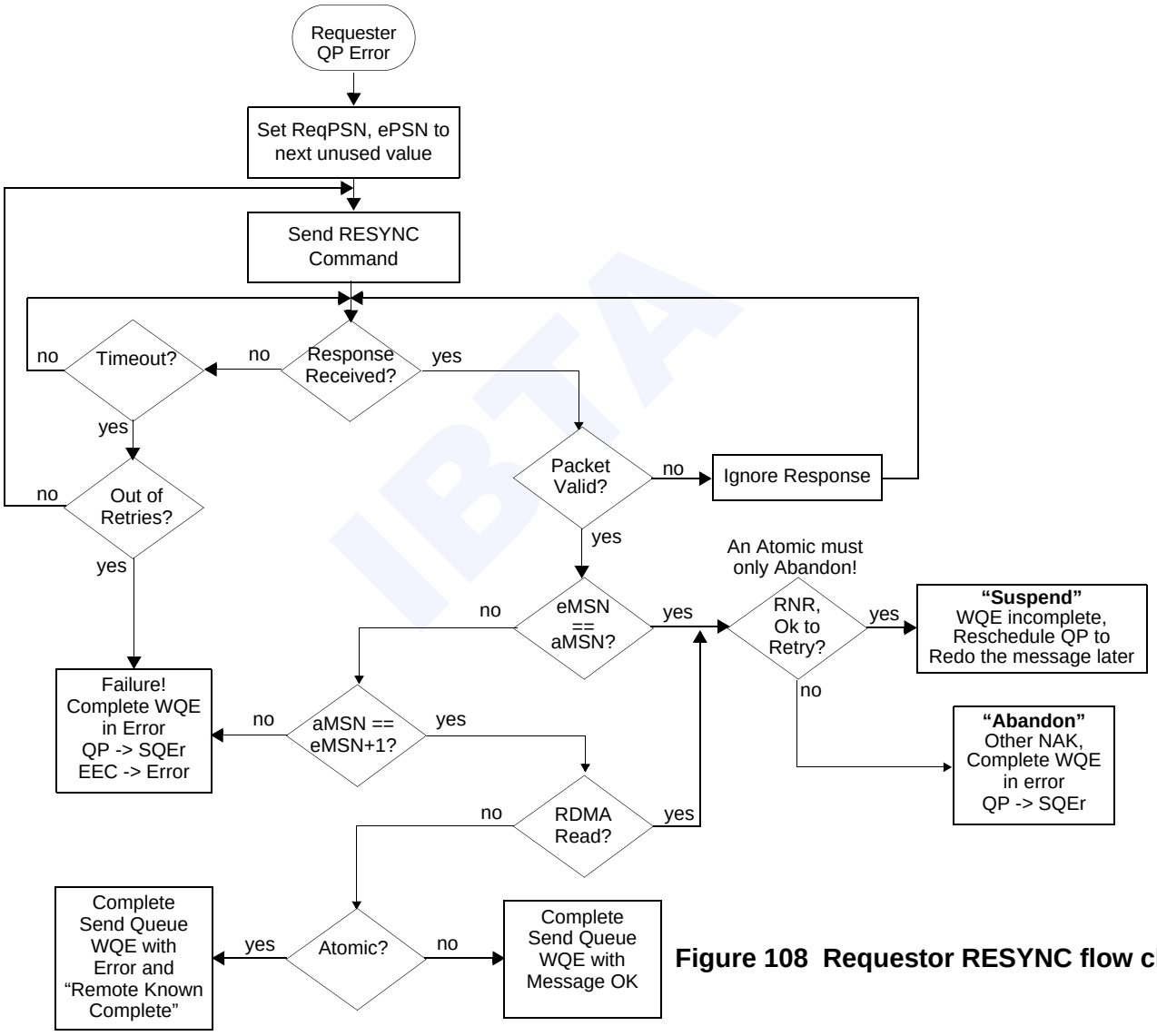
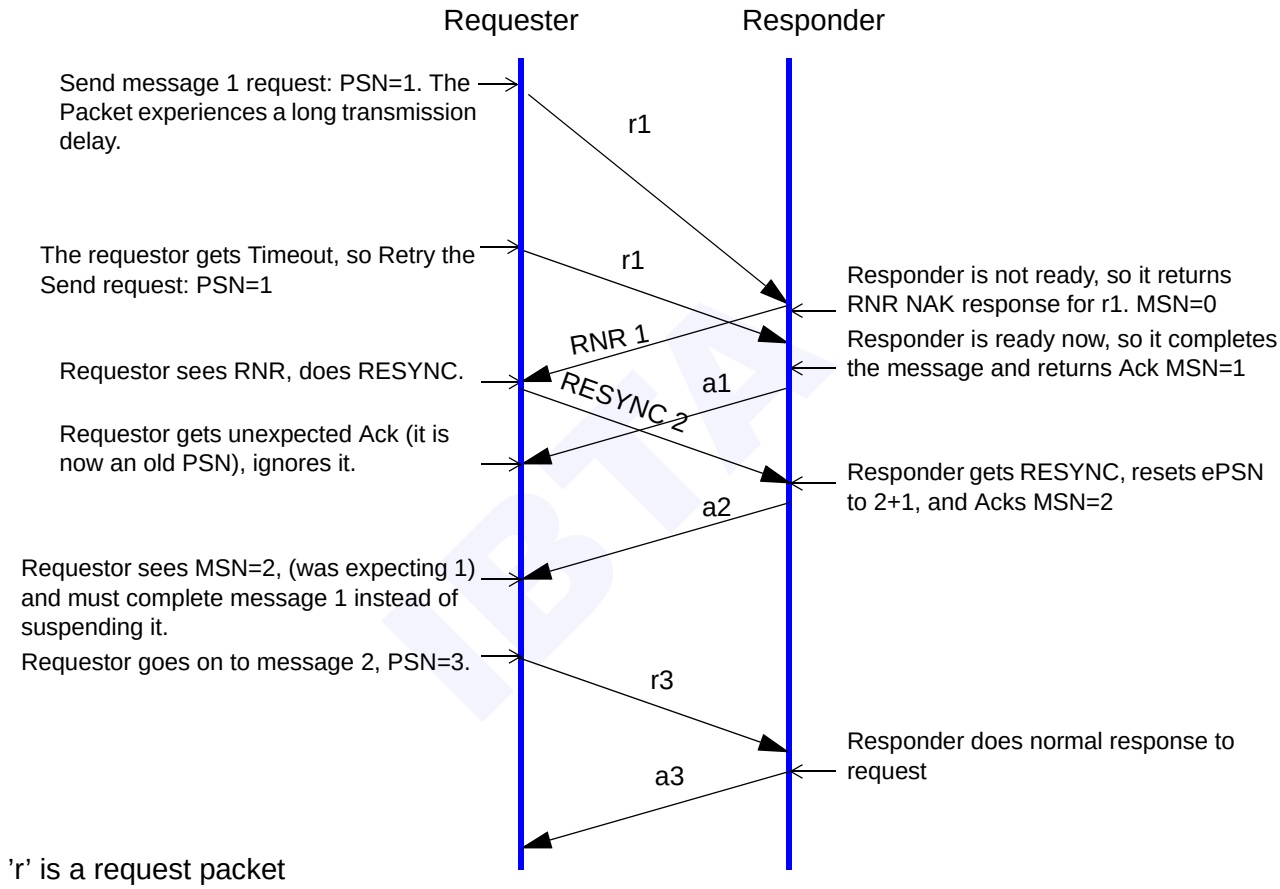


Figure 108 Requestor RESYNC flow chart

The following ladder diagram illustrates a RESYNC operation to correct the problem described in Figure 107. The RESYNC process allows getting both ends to agree on a appropriate PSN, to determine the status of the aborted message at the responder, and to inform the responder to abort a message in progress, if that is the case.

.



'r' is a request packet
'a' is an acknowledge packet (message)

Figure 109 RESYNC detects unexpectedly complete message

Another problem that RESYNC corrects is dealing with “Ghost” packets. This is illustrated in where a multi-packet message has an error on an early packet, the requestor puts the Send Queue in SQEr, and the upper layer eventually returns the Send Queue to RTS. Meanwhile, a fabric “event” causes a packet to be extremely delayed.

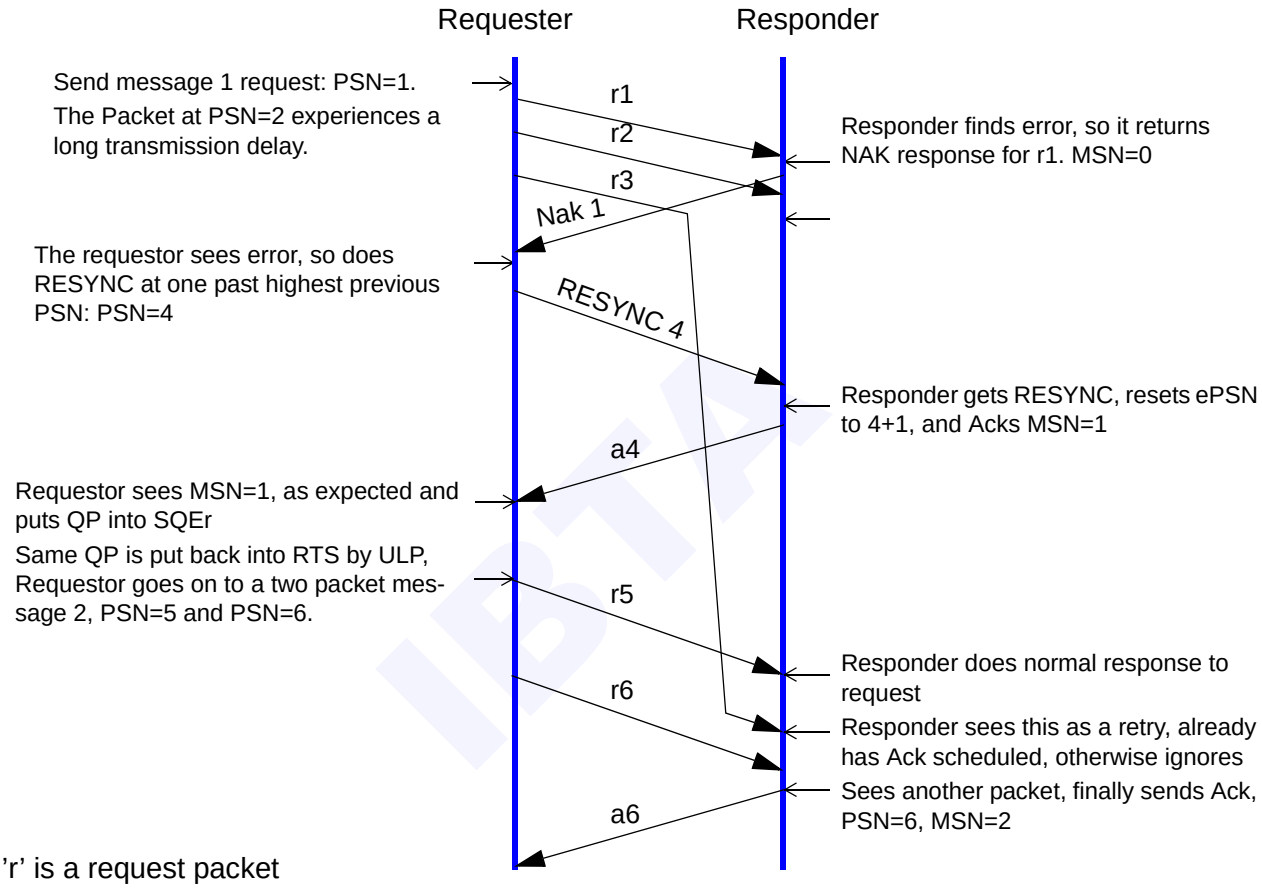


Figure 110 RESYNC prevents corruption by delayed packets

In the figure above,’r3’, being extremely delayed, arrives at the responder during packet processing of a message for the same QPs. In this case, the responder must see’r3’ as a retried packet. As such it is ignored, except to schedule an Ack, at least for Sends and RDMA Writes. For RDMA reads, the responder must create an explicit response. If a correct RDMA read was already in progress, it must be interrupted and a different response generated. This will create ghost responses at the requestor, and cause it to timeout on its RDMA Read request, with a retry to make a recovery. For an atomic, the response also appears as a ghost, or unexpected response at the requestor. In either case, it is dropped and the operation recovers.

If the RESYNC had not been performed, message 2 would have started with PSN=2. The reception of'r3' at PSN=3 would have created a data corruption problem.

o9-114.a1: For CA's claiming to support RD mode, when a message in RD mode incurs a QP related error the requestor may either:

1) Transition the QP and EEC to the error state and complete the message in error, or
2) Implement the RESYNC process as described in 9.7.8.5 Handling QP errors - RESYNC on page 368.

The RESYNC request generation is described in 9.7.3.2.2 RESYNC Generation on page 291.

o9-114.a2: For CA's that support the RD transport service, following the sending of a RESYNC, the requestor shall wait for an Ack at the RESYNC PSN. No other response is valid.

As usual, the AckReq bit should be set to insure that the responder schedules a response.

RESYNC should be timed out and retried with the usual retry count if no correct response arrives as described in 9.7.6.1.3 Detecting Lost Acknowledge Messages and Timeouts on page 338.

As usual, the requestor updates the request PSN by one prior to sending another message from the EEC.

The RESYNC response may actually complete two messages, the previous message, and the "RESYNC" message.

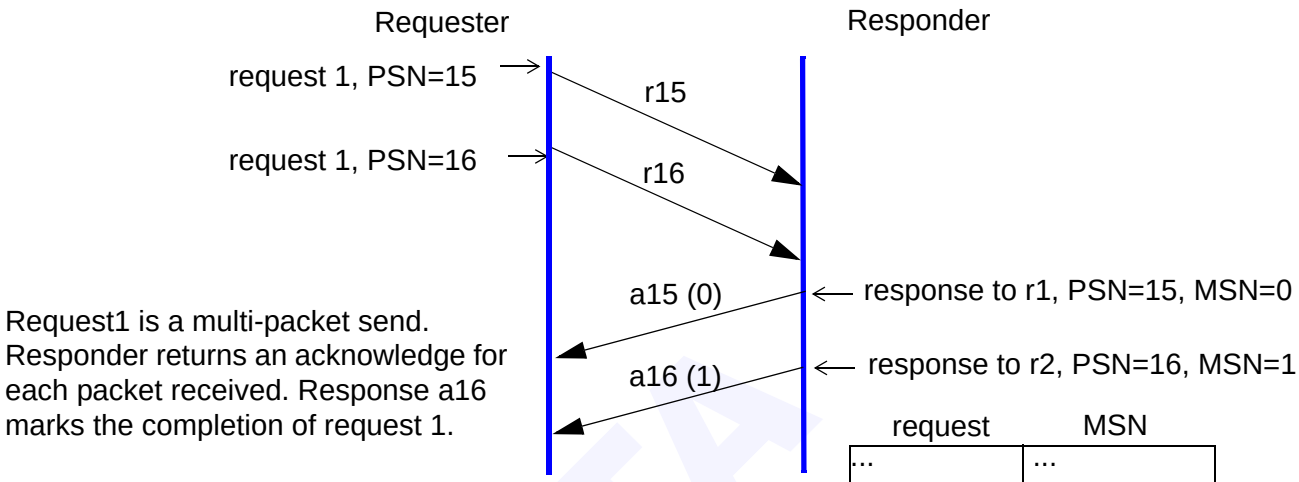
9.7.8.6 RESPONDER GENERATION OF MSN

For Reliable Datagram service, the Message Sequence Number is a number returned by the responder to the requester indicating the number of messages completed by the responder at the EE context. The MSN is carried in the three least significant bytes of the AETH. The MSN assists the requester in completing WQEs by informing the requester of the messages that have been completed by the responder.

o9-114.a3: The Responder in CAs that implement Reliable Datagram service shall return an MSN in the AETH of every response packet.

o9-114.a4: A CA responder using Reliable Datagram service shall initialize its MSN value to zero. The responder shall increment its MSN whenever it has successfully completed processing a new, valid request message. The MSN shall not be incremented for duplicate requests. The

incremented MSN shall be returned in the last or only packet of an RDMA READ or Atomic response.



request	MSN
...	...
...	.00 00 00
request 1	00 00 01
request 2	00 00 02
request 3	00 00 03
...	...

Responder's MSN

Figure 111 Responder use of MSN

9.7.8.6.1 REQUESTER BEHAVIOR ON RECEIVING A NEW MSN

The existence of a new MSN value in a response packet may be used by the requester as a signal to complete a message.

The MSN in the response to the "RESYNC" message may also be used to determine the completion status of the previous message:

- If, due to earlier retries, the previous message was actually completed, the MSN will be two higher than the last value, which was returned in the NAK response.
  - If the previous message was a Send or RDMA Write, it should be completed normally (meaning that it was not abandoned or suspended after all).
  - If the previous message was an RDMA Read, the requestor did not get the return data, so must still either suspend or abandon the message, depending on the error type.

- If the previous message was an Atomic, it must be completed in Error, the additional information “Known Complete at Responder” may be returned to the upper layers.

- If the MSN was as expected (one greater than last value), the previous message should be treated appropriately:
  - Completed in error if it is abandoned, the additional information that the message was “Known incomplete at Responder”
  - Not started if it is a Send, or RDMA Write with immediate, to be retried later
  - Incomplete if it is an RDMA Read or RDMA Write without immediate to be restarted as a new message later

If the MSN was any other value, the response is corrupted, and the EEC should be put into the error state.

9.8 UNRELIABLE SERVICE

IBA defines two types of unreliable service: Unreliable Connection (SEND, RDMA WRITE) and Unreliable Datagram (SEND only). These services have the following characteristics:

1) Requester receives no acknowledgment of message receipt
2) No packet order guarantees
3) Responder validates incoming packets as normal (validates appropriate header fields, CRC checks). A corrupted packet may be silently dropped, causing the message to be dropped.
4) On detecting an error in an incoming packet such as a dropped / out of order packet, the responder does not stop, but continues to receive incoming packets.
5) Responder considers the operation complete once it has received a complete message in correct sequence, all data has been committed to the local fault zone, and all appropriate validity checks (including variant and invariant CRC checks) have been completed. For Unreliable Connected service, the definition for a completed message is given in section 9.8.2.2.7 on page 389
6) Requester considers a message operation complete once the “last” or “only” packet has been committed to the fabric.

9.8.1 VALIDATING AND EXECUTING REQUESTS

This section applies to both unreliable connection and unreliable datagram services. Where there are differences between the services, those differences are noted. The major differences between the two services are due to the fact that Unreliable Datagram service is restricted to single packet messages whereas Unreliable Connected service does not have

this restriction. In addition, Unreliable Datagram service is restricted to using the Send function further simplifying the request validation process.

The following describes the requirements placed on a responder for validating an inbound request packet.

C9-165: The responder shall validate the various fields of the headers in order to verify the integrity of the packet. This validation process is specified in Section 9.6 Packet Transport Header Validation on page 269. Packets containing invalid fields shall be silently dropped by the responder.

C9-166: For an HCA using Unreliable Connected service, the PSN shall be examined to detect out of order packets. By examining the PSN, the responder can determine whether the packet is a new request or an invalid packet. See Section 9.8.2.2.1 Responder - Validating the PSN on page 382 for a description of this check.

o9-115: If a TCA implements Unreliable Connected service, the PSN shall be examined to detect out of order packets. By examining the PSN, the responder can determine whether the packet is a new request or an invalid packet. See Section 9.8.2.2.1 Responder - Validating the PSN on page 382 for a description of this check.

C9-167: For an HCA using Unreliable Connected service, the responder shall examine the packet OpCode to determine that the packet OpCode sequence is valid. This check is not applicable to Unreliable Datagram since that service is restricted to single packet messages, thus the concept of a sequence of opcodes is not applicable.

o9-116: If a TCA implements Unreliable Connected service, the responder shall examine the packet OpCode to determine that the packet OpCode sequence is valid. This check is not applicable to Unreliable Datagram since that service is restricted to single packet messages, thus the concept of a sequence of opcodes is not applicable.

C9-168: The responder shall examine the packet OpCode to determine whether the requested operation is supported by this receive queue.

C9-169: The responder shall verify that it has sufficient resources available to receive the message. The necessary resources include a valid receive WQE (for a SEND or an RDMA Write with immediate data), and, for a SEND request, sufficient buffer space available to receive the request.

C9-170: For an HCA responder using Unreliable Connection service, if the request is for an RDMA WRITE operation, the responder shall examine the R_Key. If the packet is found to be valid, in order, and sufficient

resources are available, it is executed by the responder. In the process of execution, the responder may encounter local errors.

o9-117: If a TCA responder implements Unreliable Connection service, and if it supports RDMA operations, it shall behave as follows. If an inbound request is for an RDMA WRITE operation, the responder shall examine the R_Key. If the packet is found to be valid, in order, and sufficient resources are available, it is executed by the responder. In the process of execution, the responder may encounter local errors.

C9-171: For an HCA responder using Unreliable Connection or Unreliable Datagram services, or for a TCA responder using Unreliable Datagram service, the responder shall follow the sequence shown in Figure 112 when validating an inbound request packet.

o9-118: If a TCA responder implements Unreliable Connection service, the responder shall follow the sequence shown in Figure 112 when validating an inbound request packet.

For Unreliable Connected service, these requirements are discussed in some detail in section 9.8.2.2 Responder Behavior on page 382. Packet validation for Unreliable Datagram service is discussed in 9.8.3.2 Responder Behavior on page 392.

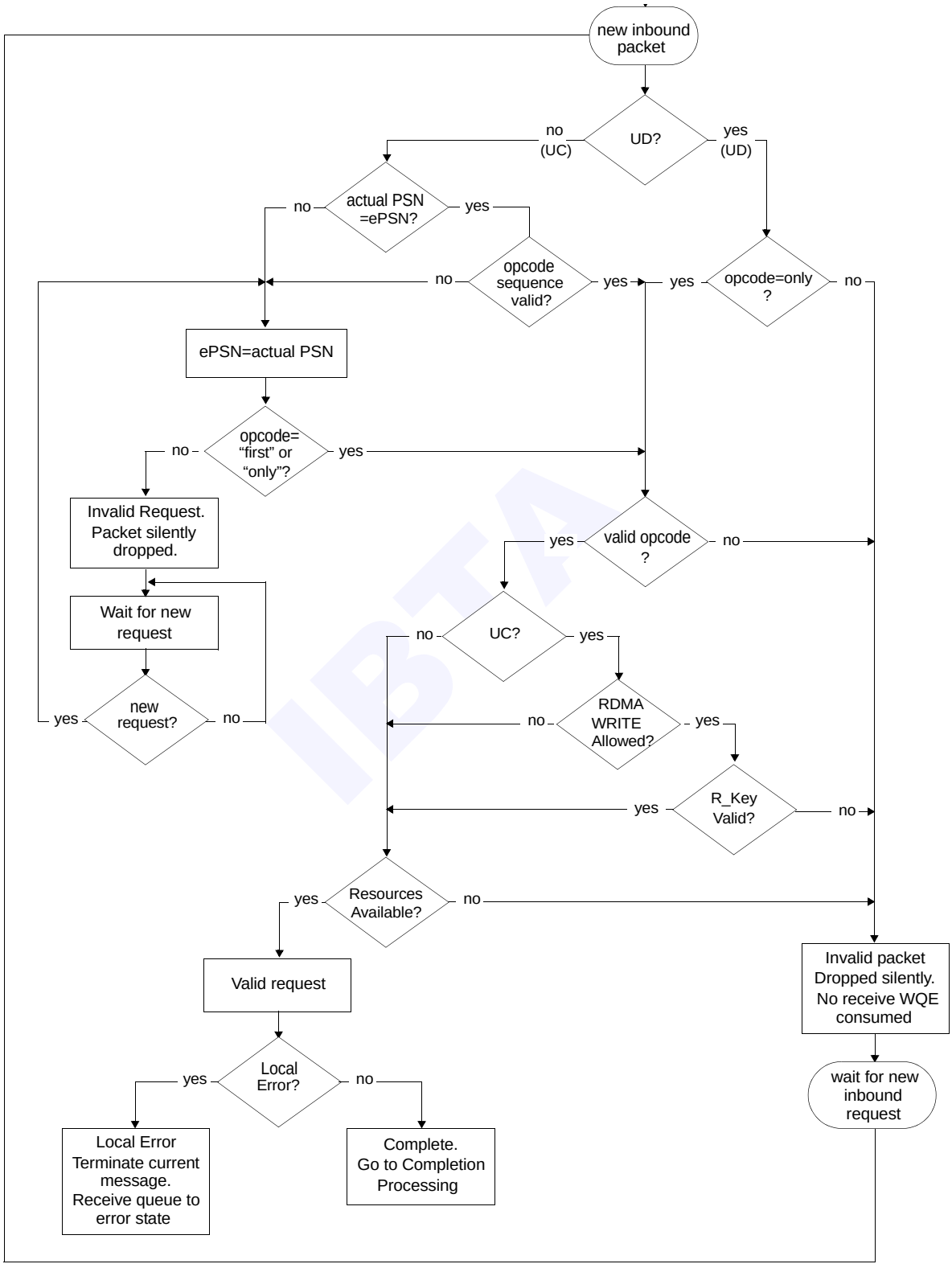


Figure 112 Unreliable Service: Inbound Packet Validation

OM10531

9.8.2 UNRELIABLE CONNECTIONS

An unreliable connection consists of a one-to-one correspondence between two QPs. Packets are sent from one QP to the other but no acknowledgments are generated by the destination QP. The chief characteristics are that there are no delivery guarantees made to the requester. The responder, however can detect data corruption and out of order packets.

The characteristics of Unreliable Connection service are summarized in Table 49.

Table 49 Summary of Unreliable Connection Service Characteristics

Characteristic	Comment
Delivery guarantee	No guarantees to the requester. Responder may drop messages.
Ordering-requester	No guarantee. Requester cannot rely on msgs arriving in order.
Ordering-responder	Responder detects and drops out of order packets.
Ordering-responder	Dropped packets may cause the message to be dropped.
Ordering-responder	After dropping a packet, responder resumes with the first packet of a new message.
Supported Operations	Sends and RDMA WRITEs (with and without Immediate data)
Message size	Maximum 2^{31} bytes. Msgs may comprise multiple packets.

9.8.2.1 REQUESTER BEHAVIOR

This section specifies the requester's required behavior when generating request packets for Unreliable Connection service.

9.8.2.1.1 REQUESTER - GENERATING PSN

C9-172: For an HCA requester using Unreliable Connection service, the requester must place a value, called the current PSN, in the BTH:PSN field of every request packet.

o9-119: If a TCA requester implements Unreliable Connection service, the requester must place a value, called the current PSN, in the BTH:PSN field of every request packet.

During connection establishment, the transport layer's client must program the next PSN to any value between zero and 16,777,215.

C9-173: For an HCA requester using Unreliable Connection service, the initial PSN, as programmed by the transport layer's client, shall appear as the BTH:PSN in the first request packet generated by the requester.

o9-120: If a TCA implements Unreliable Connection service, the initial PSN, as programmed by the transport layer's client, shall appear as the BTH:PSN in the first request packet generated by the requester.

C9-174: For an HCA using Unreliable Connection service, the transport layer shall modify (update) the PSN only when the send queue is in a proper state to transmit request packets. For example, for an HCA, the transport layer does not update the next PSN while the queue pair is in the INITIALIZED state.

o9-121: If a TCA implements Unreliable Connection service, the transport layer shall modify (update) the PSN only when the send queue is in a proper state to transmit request packets.

C9-175: For an HCA using Unreliable Connection service, each request packet generated by the requester must have a PSN value that is an increment of "1" (modulo 2^{24}) of the PSN value of the preceding request packet.

o9-122: If a TCA implements Unreliable Connection service, each request packet generated by the requester must have a PSN value that is an increment of "1" (modulo 2^{24}) of the PSN value of the preceding request packet.

Table 50 Requester's Calculation of Next PSN

Current Request Packet	PSN for Next Request Packet
SEND, RDMA WRITE	current PSN + 1 (modulo 2^{24})

9.8.2.1.2 REQUESTER - GENERATING OPCODES

The opcodes generated by a requester must fit into a schedule of opcodes as shown below.

C9-176: For an HCA requester using Unreliable Connection service, the

Table 51 Schedule of Valid OpCode Sequences

Previous Packet OpCode	Valid OpCodes for Current Packet
None e.g., first packet following connection establishment	"First" packet "Only" packet
"First" packet	"Middle" packet (message is 3 or more packets) "Last" packet (message is exactly 2 packets) Type of operation must match the previous OpCode
"Middle" packet	"Middle" packet "Last" packet Type of operation must match the previous OpCode
"Last" packet	"First" packet (1st packet of a new message) "Only" packet (1st packet of a new single packet msg)
"Only" packet	"First" packet "Only" packet

requester must generate packet opcodes which fit within the schedule of valid OpCode sequences as shown in Table 51 Schedule of Valid OpCode Sequences on page 381. When generating a request packet, the BTH:Opcode shall be as specified in Table 35 OpCode field on page 235.

o9-123: If a TCA requester implements Unreliable Connection service, the requester must generate packet opcodes which fit within the schedule of valid OpCode sequences as shown in Table 51 Schedule of Valid OpCode Sequences on page 381. When generating a request packet, the BTH:Opcode shall be as specified in Table 35 OpCode field on page 235.

9.8.2.1.3 REQUESTER - GENERATING PAYLOADS

The requester shall generate payload lengths as a function of the opcode as follows:

C9-177: For an HCA using Unreliable Connection service, if the OpCode specifies a "first" or "middle" packet, then the packet payload length must be a full PMTU size.

C9-178: For an HCA using Unreliable Connection service, if the OpCode specifies a "only" packet, then the packet payload length must be between zero and PMTU bytes in size. Thus, the only way to create a zero byte length transfer is by use of a single packet message.

C9-179: For an HCA using Unreliable Connection service, if the OpCode specifies a "last" packet, then the packet payload length must be between one and PMTU bytes in size.

o9-124: If a TCA implements Unreliable Connection service, then it shall conform to the three preceding HCA requirements for OpCode.

9.8.2.1.4 COMPLETING A MESSAGE SEND OR RDMA WRITE

C9-180: For an HCA requester using Unreliable Connection service, the requester shall consider a message Send (or RDMA WRITE) complete when either of the following conditions occurs: The requester has committed the last byte of the VCRC field of the last packet to the wire (and detected no local errors associated with the message transfer), or the requester has detected a local error associated with the message transfer that causes the requester to terminate sending the request.

o9-125: If a TCA requester implements Unreliable Connection service, the requester shall consider a message Send (or RDMA WRITE) complete when either of the following conditions occurs: The requester has committed the last byte of the VCRC field of the last packet to the wire (and detected no local errors associated with the message transfer), or the requester has detected a local error associated with the message transfer that causes the requester to terminate sending the request.

Note that at the time that the requester completes the send WQE, the state of the memory at the responder is unknown. Likewise, if the requester detects a local error while sending the request packet, the state of the responder's memory is unknown.

9.8.2.2 RESPONDER BEHAVIOR

This section specifies the responder's required behavior when receiving inbound requests.

9.8.2.2.1 RESPONDER - VALIDATING THE PSN

The responder maintains an Expected PSN value (ePSN) that it uses to detect missing packets from a multi-packet request message and to detect dropped messages. Since the PSN of every inbound request packet is sequential and monotonically increasing for UC service, a break in the PSN sequence indicates a lost or dropped request packet.

C9-181: For an HCA responder using Unreliable Connection service, the responder shall maintain an Expected PSN value (ePSN). This is the PSN that the responder expects to find in the BTH of the next inbound request packet.

o9-126: If a TCA responder implements Unreliable Connection service, the responder shall maintain an Expected PSN value (ePSN). This is the PSN that the responder expects to find in the BTH of the next inbound request packet.

The responder's expected PSN may be initialized at connection establishment time by the transport's client to any value between zero and 16,777,215. However, since the responder will accept any valid packet with an opcode of "first" or "only", and use the value of the PSN contained in such a packet as its expected PSN, it is not required that the responder's initial expected PSN be programmed. See Chapter (Chapter 12: Communication Management on page 650 for a full description of the mechanism for loading the expected PSN at connection establishment time.

The initial expected PSN can only be set by the client when the queue is in the Initialized state. Attempts by the client to set the PSN when it is in any other state may be ignored by the transport layer.

C9-182: For an HCA using Unreliable Connection service, the transport layer shall modify (update) its expected PSN only when the receive queue is in a proper state to receive inbound request packets. For example, for an HCA, the transport layer does not modify the PSN when the queue pair is in the Initialized state.

o9-127: If a TCA implements Unreliable Connection service, the transport layer shall modify (update) its expected PSN only when the receive queue is in a proper state to receive inbound request packets. For example, for an HCA, the transport layer does not modify the PSN when the queue pair is in the Initialized state.

C9-183: For an HCA responder using Unreliable Connection service, an inbound request packet shall be declared out of order if its PSN does not exactly match the responder's current ePSN.

o9-128: If a TCA responder implements Unreliable Connection service, an inbound request packet shall be declared out of order if its PSN does not exactly match the responder's current ePSN.

C9-184: An HCA responder using Unreliable Connection service shall behave as follows. If, during packet validation, an inbound request packet is discovered with an OpCode of "first" or "only", the responder shall accept the packet and shall accept the PSN of that request message as its new ePSN, regardless of whether the inbound packet is out of order or not. This shall be done regardless of the previous value of ePSN.

o9-129: A TCA responder implementing Unreliable Connection service shall behave as follows. If, during packet validation, an inbound request packet is discovered with an OpCode of "first" or "only", the responder shall accept the packet and shall accept the PSN of that request message as its new ePSN, regardless of whether the inbound packet is out of order or not. This shall be done regardless of the previous value of ePSN.

C9-185: For an HCA responder using Unreliable Connection service, before executing an inbound request, the responder shall check the PSN by comparing the PSN in the inbound BTH to the responder's expected PSN. The rules that the responder uses to calculate its next expected PSN shall be the same as those used by the requester when it calculates the PSN value to insert in its next request packet. These rules are given in 9.8.2.1.1 Requester - Generating PSN on page 379.

o9-130: For an HCA responder using Unreliable Connection service, before executing an inbound request, the responder shall check the PSN by comparing the PSN in the inbound BTH to the responder's expected PSN. The rules that the responder uses to calculate its next expected PSN shall be the same as those used by the requester when it calculates the PSN value to insert in its next request packet. These rules are given in 9.8.2.1.1 Requester - Generating PSN on page 379.

o9-131: If the PSN of the inbound message does not match the responder's ePSN, the responder may notify its client of the presence of one or more lost messages. The mechanism by which the responder notifies its client is outside the scope of this specification.

C9-186: For an HCA responder using Unreliable Connection service, if a multi-packet message is in progress at the time that an out of order packet is detected, the current message shall be silently dropped. The responder then waits for the first packet of a new message. It is possible that the present packet (the out of order packet) is the first packet of a new message. If so, it shall be treated as a new message.

o9-132: If a TCA responder implements Unreliable Connection service, if a multi-packet message is in progress at the time that an out of order packet is detected, the current message shall be silently dropped. The responder then waits for the first packet of a new message. It is possible that the present packet (the out of order packet) is the first packet of a new message. If so, it shall be treated as a new message.

A "new message" is denoted by an inbound request packet with an OpCode in the BTH of "first" or "only".

"Current message" means all the packets received since the most recently received "first" or "only" OpCode, excluding the present packet.

9.8.2.2.2 RESPONDER - OPCODE SEQUENCE CHECK

A request packet must fit within a schedule of valid OpCode sequences. The OpCode sequence is determined by examining the BTH:OpCode.

C9-187: For an HCA responder using Unreliable Connection service, the responder shall check the sequence of packet OpCodes as described in items (1) through (5) below:

1) If this is the first packet following establishment of the connection, then the packet OpCode must indicate either “first” or “only”. An OpCode of “middle” or “last” implies that at least the first packet of the current message was lost and denotes an invalid OpCode sequence.

2) If the last valid packet received had an OpCode indicating “first”, then the current OpCode must indicate either “middle” or “last”. It must also match the operation type specified in the last valid packet (SEND, RDMA WRITE). A current OpCode of “first” or “only” implies that at least the last packet of the previous message was lost and denotes an invalid OpCode sequence.

3) If the last valid packet received had an OpCode indicating “middle”, then the current OpCode must indicate either “middle” or “last”. It must also match the operation type specified in the last valid packet (SEND or RDMA WRITE request). A current OpCode of “first” or “only” implies that at least the last packet of the previous message was lost and denotes an invalid OpCode sequence.

4) If the last valid packet received had an OpCode indicating “last”, then the current OpCode must indicate either “first” or “only”. A current OpCode of “middle” or “last” implies that at least the first packet of the current message was lost and denotes an invalid OpCode sequence.

5) If the last valid packet received had an OpCode indicating “only”, then the current OpCode must indicate either “first” or “only”. A current OpCode of either “middle” or “last” implies that the first packet of the current message was missed and denotes an invalid OpCode sequence.

o9-133: If a TCA responder implements Unreliable Connection service, the responder shall check the sequence of packet OpCodes as described in items (1) through (5) above.

The responder’s behavior in the presence of an invalid OpCode sequence is specified in Section 9.9.3 Responder Side Behavior on page 408.

C9-188: For an HCA responder using Unreliable Connection service, if the responder detects an invalid OpCode sequence, the current message shall be silently dropped. The responder then waits for a new inbound request packet with an OpCode of “first” or “only”; any other inbound request packet shall be silently dropped.

o9-134: If a TCA responder implements Unreliable Connection service, and if the responder detects an invalid OpCode sequence, the current message shall be silently dropped. The responder then waits for a new inbound request packet with an OpCode of “first” or “only”; any other inbound request packet shall be silently dropped.

"Current message" means all the packets received since the most recently received "first" or "only" OpCode, excluding the present packet.

C9-189: For an HCA responder using Unreliable Connection service, if the present packet, which caused the invalid OpCode sequence, has an OpCode of "first" or "only" it shall be treated as the first packet of a new request message.

o9-135: If a TCA responder implements Unreliable Connection service, and if the present packet, which caused the invalid OpCode sequence, has an OpCode of "first" or "only" it shall be treated as the first packet of a new request message.

The list of valid OpCode sequences is summarized in the following table.

Table 52 Summary: Valid OpCode Sequences

Previous Packet OpCode	Valid OpCodes for Current Packet
None e.g., first packet following connection establishment	"First" packet "Only" packet
"First" packet	"Middle" packet (message is 3 or more packets) "Last" packet (message is exactly 2 packets) Type of operation must match the previous OpCode
"Middle" packet	"Middle" packet "Last" packet Type of operation must match the previous OpCode
"Last" packet	"First" packet (1st packet of a new message) "Only" packet (1st packet of a new single packet msg)
"Only" packet	"First" packet "Only" packet

9.8.2.2.3 RESPONDER OPCODE VALIDATION

C9-190: For UC, the responder shall validate the requested function (SEND or RDMA WRITE) is supported by the receive queue and that the BTH:OpCode is not reserved before executing the request.

Note that the OpCode was also examined as part of packet validation in section 9.6 Packet Transport Header Validation on page 269 to ensure that the inbound packet contains a request for Unreliable Connected service.

C9-191: Invalid UC requests shall be silently dropped by the responder per 9.9.3 Responder Side Behavior on page 408.

9.8.2.2.4 RESPONDER REMOTE ACCESS VALIDATION

C9-192: This compliance statement has been obsoleted and replaced by C9-192.2.1:

C9-192.2.1: For an HCA responder using Unreliable Connection service, if the inbound request is for a RDMA WRITE and the requested DMA length in the RETH is non-zero, then the following conditions shall be checked:

- The R_Key field in the RETH is valid.
- The virtual address and length is within the locally defined limits associated with the R_Key. For an RDMA WRITE request, the length check is conducted on a per packet basis and is based on the LRH:PktLen field.
- The type of access specified (Write) is within the locally defined limits associated with the R_Key.
- For an HCA, the protection domain shall be checked according to the conditions defined in Section 10.2.3 Protection Domains on page 434.

A failure of any of these checks constitutes an R_Key violation. The responder's behavior in response to an R_Key violation is specified in Section 9.9.3 Responder Side Behavior on page 408.

o9-136: If a TCA responder implements Unreliable Connection service and RDMA functionality, it shall conform to the preceding HCA compliance statement.

C9-193: For an HCA using Unreliable Connection service, the R_Key field shall not be checked for a zero-length RDMA WRITE request, even if the request includes Immediate data.

o9-137: If a TCA responder implements Unreliable Connection service and RDMA functionality, the R_Key field shall not be checked for a zero-length RDMA WRITE request, even if the request includes Immediate data.

9.8.2.2.5 RESPONDER - LENGTH VALIDATION

C9-194: For an HCA responder using Unreliable Connection service, the PktLen field of the LRH shall be checked to confirm that there is sufficient space available in the receive buffer specified by the receive WQE. This check applies only to SENDs.

o9-138: If a TCA responder implements Unreliable Connection service, the PktLen field of the LRH shall be checked to confirm that there is sufficient space available in the receive buffer specified by the receive WQE. This check applies only to SENDs.

The length of the packet shall also be validated by comparing it to the OpCode as follows:

C9-195: For an HCA responder using Unreliable Connection service, if the UC BTH:OpCode specifies a “first” or “middle” packet, then the packet payload length must be a full PMTU size.

o9-139: If a TCA responder implements Unreliable Connection service, and if the UC BTH:OpCode specifies a “first” or “middle” packet, then the packet payload length must be a full PMTU size.

C9-196: For an HCA responder using Unreliable Connection service, if the UC BTH:OpCode specifies a “only” packet, then the packet payload length must be between zero and PMTU bytes in size. Thus, the only way to create a zero byte length transfer is by use of a single packet message.

o9-140: If a TCA responder implements Unreliable Connection service, and if the UC BTH:OpCode specifies a “only” packet, then the packet payload length must be between zero and PMTU bytes in size. Thus, the only way to create a zero byte length transfer is by use of a single packet message.

C9-197: For an HCA responder using Unreliable Connection service, if the UC BTH:OpCode specifies a “last” packet, then the packet payload length must be between one and PMTU bytes in size.

o9-141: If a TCA responder implements Unreliable Connection service, and if the UC BTH:OpCode specifies a “last” packet, then the packet payload length must be between one and PMTU bytes in size.

C9-198: This compliance statement is obsolete and has been removed.

o9-142: This compliance statement is obsolete and has been removed.

C9-199: For an HCA responder using Unreliable Connection service, if the BTH:OpCode field[4:0] specifies a first or middle request packet (e.g. SEND First, or RDMA WRITE Middle), the pad count bits are verified to be b00, indicating no pad bytes are present. If the pad count bits are non-zero, the OpCode is invalid.

o9-143: If a TCA responder implements Unreliable Connection service, and if the BTH:OpCode field[4:0] specifies a first or middle request packet (e.g. SEND First, or RDMA WRITE Middle), the pad count bits are verified to be b00, indicating no pad bytes are present. If the pad count bits are non-zero, the OpCode is invalid.

If a packet is detected with an invalid length the request is an invalid request. The responder’s behavior in such a case is specified in Section 9.9.3.1 Responder Side Error Response on page 412.

9.8.2.2.6 RESPONDER - LOCAL OPERATION VALIDATION

A valid inbound request may still fail to complete due to a failure that is local to the responder, e.g. local memory translation error while accessing local memory. A local error may cause the receive queue to transition to the error state. See 9.9.3 Responder Side Behavior on page 408 for additional details.

9.8.2.2.7 COMPLETING A MESSAGE RECEIVE

The responder considers a given inbound message completed successfully when it has:

- Detected the beginning of a valid message as indicated by the presence of a “First packet” or “Only packet” OpCode in the BTH,
- Detected the end of the same valid message as indicated by the presence of a “Only packet” or “Last packet OpCode in the BTH, without a skip in the PSN sequence,
- Received all the packets between “First packet” and “Last packet” inclusive successfully and in order, or has successfully received an “Only packet”.
- Committed the message payload to the local fault zone without error, and,
- Successfully completed all appropriate validity checks (including variant and invariant CRC).

A failure detected during any of these steps may or may not cause the associated WQE to be completed in error. In some cases, such as a missing “first” packet, it is entirely likely that no WQE will be consumed by the responder. Note that, in the presence of errors, it is not possible to guarantee the state of the responder’s memory. Some or all of a given packet may have been committed to the responder’s memory before the error is detected.

Once an inbound message receive is completed successfully, the responder completes the current WQE.

9.8.3 UNRELIABLE DATAGRAMS

Unreliable Datagrams are a form of communication that allow a source QP to send each message to one of many destination QPs that may exist on the same or multiple destination endnodes.

- For each message to be sent, the requester must be supplied with the destination address (see 11.2.2.1 Create Address Handle on page 559), the destination QP, the destination Q_Key etc. See 11.4.1.1 Post Send Request on page 612 for the parameters supplied for an HCA.

- The responder must deliver to the client the requester's address, QP etc. See 11.4.2.1 Poll for Completion on page 623 for more detail on HCA requirements.

Table 53 Unreliable Datagram QP characteristics

Property / Level of Reliability	Support
Corrupt data detected and dropped	Yes, silent drop on error
OpCode Service and command Validation	Yes, silent drop on error
Receive buffer overrun	Yes, reported as WR error
Data repeated	No
Data order guaranteed	No
Data loss detected	Not required
RDMA Support	No
ATOMIC Support	No
Immediate data support	Yes
Max Size of SEND messages	PMTU-sized packet - 256 - 4096 bytes of data payload. Any message that exceeds the PMTU will not be delivered.
State of SEND when request completed	Committed to transmission on the fabric

C9-200: Devices that source UD messages shall limit the UD message size to a single packet. The packet should be no larger than the PMTU between the source and destination (or it will be dropped).

C9-201: Devices that source and sink UD messages shall meet the requirements of the basic Unreliable Services (see 9.8 Unreliable Service on page 375 through 9.8.1 Validating and Executing Requests on page 375).

C9-202: Devices that source and sink IBA UD messages shall meet the requirements specified in 9.9 Error detection and handling on page 396.

This figure shows two views of Unreliable Datagram QPs. Process A on Processor 1 communicates with three processes: processes C and D on Processor 2 and process E on processor 3.

The view on the right shows how software might view the connection. Buffers in the Send Q flow into buffers in the Receive Queue on the connected QP.

The lower view gives a hardware centric view, showing some of the state maintained per connected QP. Since each QP is connected to nothing, the destination and other necessary information (SL, DGID etc.) is picked with each message. The PSNs, while generated, are not checked.

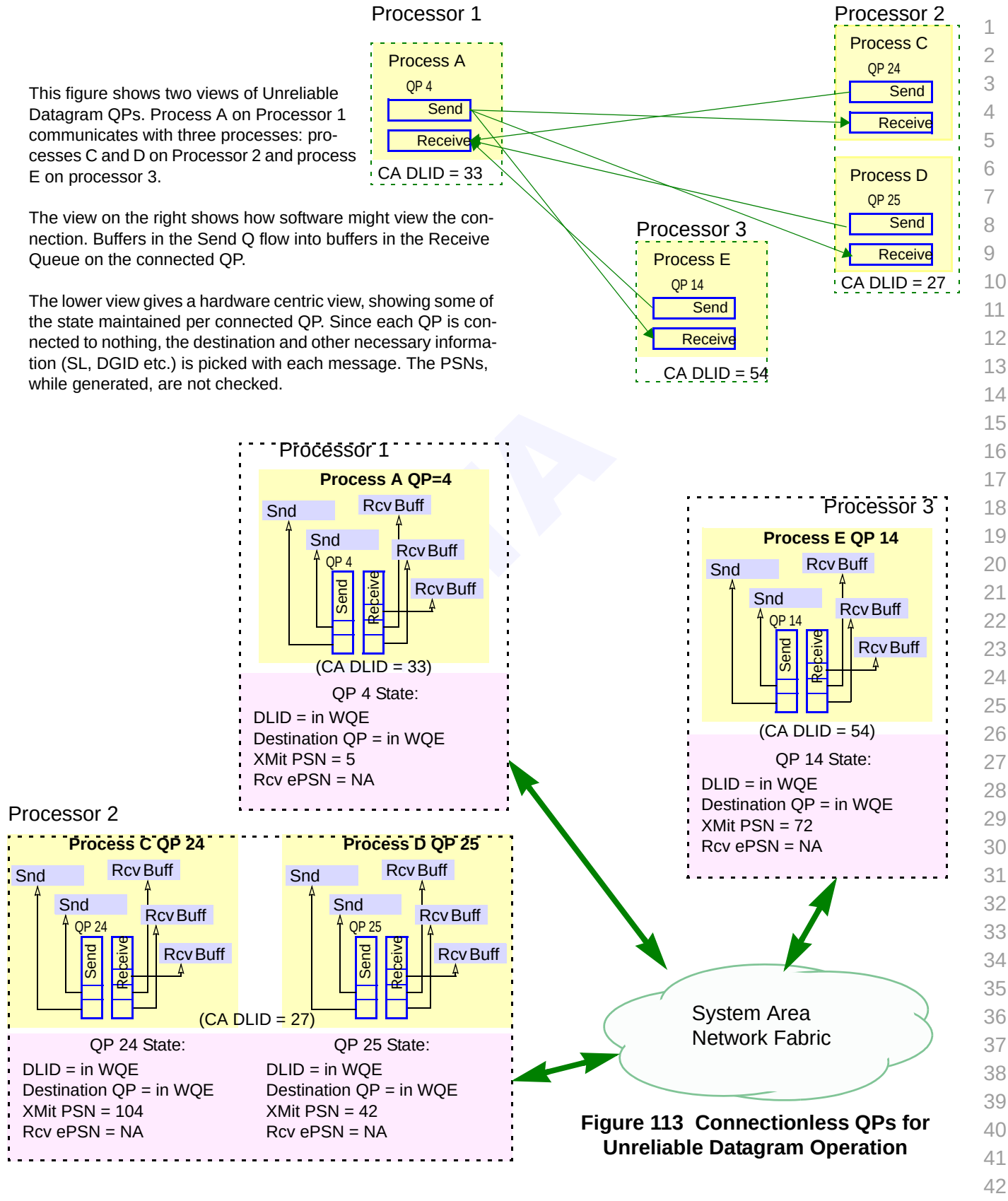


Figure 113 Connectionless QPs for Unreliable Datagram Operation

9.8.3.1 REQUESTER BEHAVIOR

This section specifies the requester's required behavior when generating request packets.

C9-203: Devices that source UD messages shall meet the requirements specified in 9.8.3.1.1 Generating PSN on page 392 and 9.8.3.1.2 Completing a Message Send on page 392 while sending UD messages.

9.8.3.1.1 GENERATING PSN

C9-204: For each request message on a UD transport service, the requester shall generate PSNs that is an increment of "1" (modulo 2^{24}) of the PSN value of the preceding request packet. The incrementing of PSN is not required for QP0 and QP1.

The initial PSN value shall be loaded by the transport's client while the send queue is in the Initialized state and may be initialized to any 24-bit value. While in the process of transmitting request packets, the transport layer shall modify (update) the PSN only when the send queue is in the Ready to Send state.

9.8.3.1.2 COMPLETING A MESSAGE SEND

The requester shall consider a message Send complete when it has:

- Committed the last byte of the VCRC field of the packet to the wire, and detected no local errors associated with the message transfer.
- Detected a local error associated with the message transfer that causes the requester to terminate sending the request.

Note that at the time that the requester completes the send WQE, the state of the memory at the responder is unknown. Likewise, if the requester detects a local error while sending the request packet, the state of the responder's memory is unknown.

9.8.3.2 RESPONDER BEHAVIOR

This section specifies the responder's required behavior when receiving inbound requests.

9.8.3.2.1 RESPONDER - VALIDATING THE PSN

o9-144: For UD transport service, the responder may ignore the PSN field.

Some applications (e.g. multicast-based media streaming) may derive benefit from having the responder validate the PSN sequence to detect out-of-sequence packets. It is permissible for a responder implementation to do so, but is outside the scope of the IBA specification.

9.8.3.2.2 RESPONDER - LENGTH VALIDATION

C9-205: Before executing the request, the responder shall validate the Packet Length field of the LRH and the PadCnt of the BTH as described in 9.8.3.2.2: Responder - Length Validation.

The following characteristics shall be validated:

- The Length fields shall be checked to confirm that there is sufficient space available in the receive buffer specified by the receive WQE.
- The packet payload length must be between zero and PMTU bytes inclusive in size.

If a packet is detected with an invalid length, the request shall be an invalid request and it shall be silently dropped by the responder as specified in Section 9.9.3 Responder Side Behavior on page 408. The responder then waits for a new request packet.

9.8.3.2.3 RESPONDER OPCODE VALIDATION

C9-206: For UD, the responder shall validate the BTH:OpCode for the requested function (SEND) is supported by this receive queue and is not reserved before executing the request else the request is invalid.

C9-207: If a UD receive queue does not have an entry to hold an inbound SEND request, the request is invalid.

If the request is invalid, it shall be silently dropped by the responder as specified in Section 9.9.3 Responder Side Behavior on page 408.

9.8.3.2.4 RESPONDER - LOCAL OPERATION VALIDATION

A valid inbound request may still fail to complete due to a failure that is local to the responder, e.g. local memory translation error while accessing local memory. A local error may cause the receive queue to transition to the error state. See 9.9.3 Responder Side Behavior on page 408 for additional details.

9.8.3.2.5 COMPLETING A MESSAGE RECEIVE

The responder considers a given inbound message completed successfully when it has:

- Committed the message payload to the local fault zone without error
- Successfully completed all appropriate validity checks (including variant and invariant CRC).

A failure detected during any of these steps may or may not cause the associated WQE to be completed in error. In some cases, such as an opcode or length error, no WQE will be consumed by the responder. Note that, in the presence of errors, it is not possible to guarantee the state of the responder's memory. Some or all of a given packet may have been

committed to the responder's memory before the error is detected. Once an inbound message receive is completed successfully, the responder completes the current WQE.

9.8.4 RAW DATAGRAMS

The previous several sections describe the different transport protocols defined by the IBA specification. In addition to these, IBA allows other protocols to be carried by an IBA subnet. IBA datagrams that encapsulate such traffic are referred to as Raw Datagrams.

IBA defines two different methods to support Raw Datagrams. In Section 7.7.5 Link Next Header (LNH) - 2 bits on page 194 two bits in the local route header are used to specify the next header after the LRH. The following table describes the two LNH encodings that describe Raw Datagrams.

Link Next Header LNH(1:0)	
IBA_Transport	**GRH (IPv6) header**
0	1
0	0

Structure of the Raw Datagram

LRH	IPv6	Packet Payload	VCRC
LRH	RWH	Packet Payload	VCRC

Figure 114 Raw Datagrams

The first method of encoding a Raw Datagram is used only for IPv6 datagrams. The packet payload may contain any transport or network protocol defined by the IETF's encoding of the IPv6 header's "next header" field excluding any encoding indicating the next header is an IBA transport header.

C9-208: CAs shall not generate an outbound packet and will discard any inbound packet whose LRH indicates a Raw Datagram and whose IPv6 "next header" indicates an IBA transport. TCAs may report this error in any manner they choose.

The second method of encoding a Raw Datagram uses the IBA defined raw header (RWH). The RWH contains the 16-bit Ethertype field - the RWH is described in section 5.3 Raw Packet Format on page 161. The RWH is used to define the protocol header encapsulated in the packet payload. In general, the second method is used to allow protocols not supported by the IPv6 next header - it should be noted that either method may be used to transport IPv6 datagrams.

o9-145: If a CA implements Raw Datagram support, the Packet Payload of Raw Datagrams must always be of a modulo 4 size, since the LRH Packet length describes the length in 4 byte increments. Should the encapsulated payload size not be a multiple of 4 bytes, the payload shall be padded to a multiple of 4 bytes.

o9-146: If a CA implements Raw Datagram support, the QPs used to inject and consume Raw Datagrams shall be locally managed, i.e. the association of a QP with a given Raw Datagram service is implementation dependent.

o9-147: If a CA implements Raw Datagram support, it may support one or more QPs for Raw Datagram operations.

9.8.4.1 RAW DATAGRAM PACKET SIZE

The IBA MTU defines the maximum size of an IBA transport's data payload. The maximum size of an IBA packet is MTU+124 bytes[12] (see 7.7.8 Packet Length (PktLen) - 11 bits on page 194).

o9-148: If a CA implements Raw Datagram support, and since a Raw datagram does not use IBA transport headers, raw datagrams may have a packet payload larger than the supported MTU (see Figure 114 Raw Datagrams on page 394). The table below summarizes the maximum packet payload (and the corresponding value for the LRH PktLen field) for each of the two raw datagram types.

Table 54 Maximum Raw Datagram Packet Payload

MTU	IPv6 Raw Datagram		RWH Raw Datagram	
	Largest Possible Packet Payload[a]	Corresponding PktLen Value	Largest Possible Packet Payload[b]	Corresponding PktLen Value
256	332 Bytes	95	368 Bytes	95
512	588 Bytes	159	624 Bytes	159
1024	1100 Bytes	287	1136 Bytes	287
2048	2124 Bytes	543	2160 Bytes	543
4096	4172 Bytes	1055	4208 Bytes	1055

a. largest possible IPv6 raw packet payload = MTU + 124 (the largest packet header/CRC size) - 8 (LRH size) - 40 (IPv6 header size)
b. largest possible RWH raw packet payload = MTU + 124 (the largest packet header/CRC size) - 8 (LRH size) - 4 (RWH header size)

12. The 124B maximum packet header/CRC byte count does not include the VCRC field.

9.9 ERROR DETECTION AND HANDLING

IBA uses a layered error management architecture (LEMA) approach. Each level is responsible for detecting and managing errors appropriate to that layer before passing the packet or message up to the next layer in the stack.

Thus the transport layer responds to errors particular to the transport including errors in the packet header and failures to correctly transport a message.

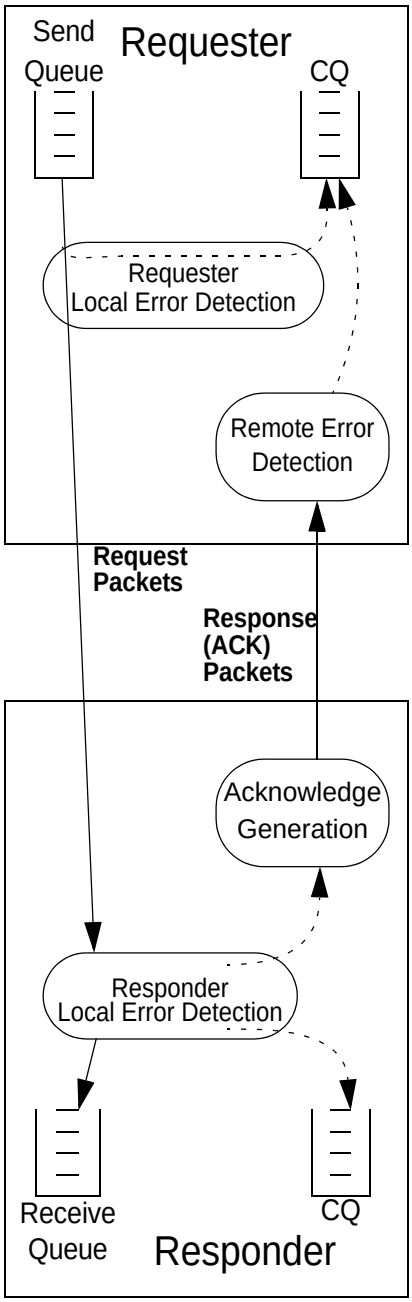


Figure 115 Requester / Responder Error Detection

Errors detected in the transport layer are reported to the transport's client. In this section, the interface between the transport layer and its client is shown conceptually as the send or Receive Queue. In the case of an HCA, the transport indicates errors to its client by writing a completion code to a Completion Queue Entry (CQE) on the Completion Queue (CQ). As usual TCAs are free to report errors (or not) as they see fit.

In order to simplify the discussion, error behavior is discussed separately for the requester and responder ends. This causes a slight amount of duplication between the summary tables in the following sections describing the errors for the requester and responder side. Specifically, overlaps occur when an error is detected by the responder and reported to the requester. These areas of overlap, however, are strictly confined to reliable classes of service.

Errors that are reported by the requester to its client fall into one of two classes. The first are Locally Detected errors; i.e., errors that are detected solely by the requester side. An example of a locally detected error is a protection fault detected by the requester while accessing its own local memory during a send request.

The second class is remotely detected errors, which are those errors detected by the responder and reported to the requester via a NAK syndrome in an Response packet. Remotely detected errors only apply to the reliable classes of service (reliable connected and reliable datagram).

Whereas there were two classes of errors for the requester side (locally and remotely detected), there are only locally detected errors on the responder side.

In response to a locally detected error, the responder side may be required to report the error to the requester, or it may be required to report the error to its local client, or both, or neither. The choice of to whom the error is reported is governed by the class of service (reliable versus unreliable), and the specific error that is detected.

The key focus of the following sections is to categorize all errors according to how errors are reported to the transport layer's client, and the behavior that the send (receive) queue must exhibit following detection of an error. Thus, this section is categorized according to not only where an error is detected, but to whom it is reported.

9.9.1 REPORTING ERRORS TO THE VERBS LAYER

For an HCA, the IBA software interface defines three types of errors that can be reported through the verbs layer. These are called immediate errors, completion errors, and asynchronous errors. Of those three types, the transport layer is only capable of reporting completion errors or asynchronous errors. This is because immediate errors are detected by the verbs layer before the WQE ever gets posted to the transport layer. Table 55 summarizes the types of errors that an IBA transport can detect and report to the verbs layer. For more information on these error types, see 10.10.2 Error Handling Mechanisms on page 530.

Table 55 Software Error Types Detected by Transport Layer

IBA Software Defined Error Types	Detected by IBA Transport
Immediate Errors	no
Completion Errors - Interface check	yes
Completion Errors - Processing error	yes
Asynchronous Errors - Affiliated type	yes
Asynchronous Errors - Unaffiliated type	yes

There are two classes of completion errors: Interface checks and processing errors. An interface check is an error in the information supplied to the Channel Interface detected before data is placed onto the link. A processing error is an error encountered during the processing of the work request by the Channel Interface.

9.9.2 REQUESTER SIDE ERROR BEHAVIOR

As indicated above, the requester detects errors originating locally or remotely.

9.9.2.1 REQUESTER SIDE ERROR DETECTION - LOCALLY DETECTED ERRORS

A locally detected error reflects either an error condition that has occurred in the requester's channel interface, a missing response from the responder side (timeout) or excessive retries for sequence errors or RNR NAKs.

Locally detected errors at the requester can occur during request packet generation, during the processing of response packets, or due to a timeout.

C9-209: For an HCA requester using RC, UC, or UD service, and for a TCA requester using UD service, the requester shall behave as follows. For locally detected transport errors that are detected during transmission of request packets, a CA shall stop transmission for the affected QP, shall store the state associated with the error until any previous incomplete WQEs are completed, and finally complete the affected WQE. The QP shall be put into the error state if the error type requires this.

o9-149: If a TCA requester implements Reliable Connection or Unreliable Connection service, it shall behave as follows. For locally detected transport errors that are detected during transmission of request packets, a CA shall stop transmission for the affected QP, shall store the state associated with the error until any previous incomplete WQEs are completed, and finally complete the affected WQE. The QP shall be put into the error state if the error type requires this.

o9-150: If a CA requester implements Reliable Datagram service, it shall behave as follows. For locally detected transport errors that are detected during transmission of request packets, a CA shall stop transmission for the affected EEC, shall store the state associated with the error until any previous incomplete WQEs are completed, and finally complete the affected WQE. The EEC shall be put into the error state if the error type requires this.

This is required to maintain the ordered completion of WQEs and to ensure that the error is properly reported in the WQE where the error occurred.

9.9.2.1.1 REQUESTER ERROR RETRY COUNTERS

C9-210: For an HCA using Reliable Connection service, in order to detect excessive retries, the requester shall maintain the RNR NAK and Error retry counters that perform the logical functions described in 9.9.2.1.1 Requester Error Retry Counters on page 398.

o9-151: If a CA requester implements Reliable Datagram service, or if a TCA requester implements Reliable Connection service, the requester shall behave as follows. In order to detect excessive retries, the requester shall maintain the RNR NAK and Error retry counters that perform the logical functions described in 9.9.2.1.1 Requester Error Retry Counters on page 398.

Implementations may implement these retry counters in any way they choose, but for clarity, they are here described as down counters, initialized to the number of retries allowed before terminating the operation and creating the final completion error. See 10.2.4.3 Modifying Queue Pair Attributes on page 437 for the programming of these counters in an HCA.

The RNR NAK retry counter is decremented each time the responder returns an RNR NAK. If the requester's RNR NAK retry counter is zero, and an RNR NAK packet is received, an RNR NAK retry error occurs. Each time an RNR NAK is cleared (i.e., an acknowledge message other than an RNR NAK is returned), the retry counter is reloaded. An exception to the following is if the RNR NAK retry counter is set to 7. This value indicates infinite retry and the counter is not decremented.

The Error retry counter is decremented each time the requester must retry a packet due to a Local Ack Timeout, NAK-Sequence Error, or Implied NAK. If the requester's retry counter decrements to zero, one of two things may be implemented.

If Automatic Path migration is not supported, or has already been completed, a "Transport Retry Counter Exceeded" error shall be reported in the completion.

o9-152: If a CA supports Automatic Path Migration, then, following a potentially recoverable error and its retries, the requester may migrate the connection or EE context and perform the Error retries again before finally reporting the completion in error. See 17.2.8 Automatic Path Migration on page 1031 for more information.

Each time a packet is properly acknowledged, the retry counter shall be reloaded.

9.9.2.2 REQUESTER SIDE ERROR DETECTION - REMOTELY DETECTED ERRORS

A remotely detected error occurs when the responder reports an error to the requester. Remotely detected errors are unique to reliable classes of service.

Remotely detected errors are reported via a NAK code carried in an acknowledge message. However, not all NAK codes result in an error being reported to the requester's client.

Of the possible NAK codes, two (NAK-Sequence Error and NAK-RNR) indicate operations that should be retried automatically by the requester.

The NAK codes other than NAK-Sequence Error and NAK-RNR indicate failures that must be reported to the requester's client immediately and cannot be retried.

9.9.2.3 SUMMARY - REQUESTER SIDE ERROR BEHAVIOR

Table 56 lists all errors that are detected by the requester, including both locally and remotely detected errors. If the error is detected locally by the requester, the column labelled "Syndrome" contains the notation "locally detected error". If the error is detected remotely by the responder, this

column lists the NAK syndrome that was returned by the responder. The fault behavior class specifies the actions that the requester takes to report the error to its client.

Each fault behavior class is specified below in Sections 9.9.2.4.1 through 9.9.2.4.5. For convenience, the six possible classes of fault behaviors are summarized in Table 57 Summary of Requester Fault Behavior Classes on page 403 below.

C9-211: This compliance statement is obsolete and has been replaced by C9-211.1.1:.

C9-211.1.1: For the implemented subset of transport services, requesters shall conform to the error behavior as specified in Table 56. Also, the requester's send queue shall behave as specified for each Fault Behavior Class shown in Table 57 and each of the Requester Class Fault descriptions.

Table 56 Requester Side Error Behavior

Error	Description	Syndrome	Requestor Fault Behavior Class
Packet sequence error. Retry limit not exceeded.	Responder detected a PSN larger than it expected. Requester may retry the request.	NAK-Sequence Error	RC: Class A RD: Class A else: NA
Packet sequence error. Retry limit exceeded.	Responder detected a PSN larger than it expected. The requestor performed retries, and automatic path migration and additional retries, if applicable, but all attempts failed.	NAK-Sequence Error	RC: Class B RD: Class D else: NA
Implied NAK sequence error. Retry limit not exceeded.	Requestor detected an ACK with a PSN larger than the expected PSN for an RDMA READ or ATOMIC response. Requester may retry the request.	locally detected error	RC: Class A RD: Class A else: NA
Implied NAK sequence error. Retry limit exceeded.	Requestor detected an ACK with a PSN larger than the expected PSN for an RDMA READ or atomic response. The requestor performed retries, and automatic path migration and additional retries, if applicable, but all attempts failed.	locally detected error	RC: Class B RD: Class D else: NA
Local Ack Timeout error. Retry limit not exceeded.	No ACK response from responder within timer interval. Requester may retry the request.	locally detected error	RC: Class A RD: Class A else: NA
Local Ack Timeout error. Retry limit exceeded.	No ACK response within timer interval. The requestor performed retries, and automatic path migration and additional retries, but all attempts failed.	locally detected error	RC: Class B RD: Class D else: NA
RNR NAK Retry error. Retry limit not exceeded.	Responder returned RNR NAK. Requestor may retry the request.	RNR-NAK	RC: Class A RD: Class A else: NA
RNR NAK Retry error. Retry limit exceeded.	Excessive RNR NAKs returned by the responder. Requestor retried the request “n” times, but received RNR NAK each time.	locally detected error	RC: Class B RD: Class B else: NA
Unsupported OpCode.	Responder detected an unsupported OpCode.	NAK-Invalid Request	RC: Class B RD: Class B else: NA
Unexpected OpCode.	Responder detected an error in the sequence of OpCodes, such as a missing “Last” packet. Note: there is no PSN error, thus this does not indicate a dropped packet.	NAK-Invalid Request	RC: Class B RD: Class B else: NA
Local Memory Protection Error.	Requester detected an implementation specific memory protection error in its local memory subsystem.	locally detected error	All: Class B

Table 56 Requester Side Error Behavior (Continued)

Error	Description	Syndrome	Requestor Fault Behavior Class
R_Key Violation	Responder detected an invalid R_Key while executing an RDMA Request	NAK-Remote Access Error	RC: Class B RD: Class B else: NA
Remote Operation Error	Responder encountered an error, (local to the responder), which prevented it from completing the request.	NAK-Remote Operation Error	RC: Class B RD: Class B else NA
Local Operation Error[a] - WQE	An error occurred in the requester's local channel interface that can be associated with a certain WQE.	locally detected error	All: Class B
Local Operation Error[a] - affiliated or unaffiliated	An error occurred in the requester's local channel interface that cannot be associated with a certain WQE.	locally detected error	All: Class C
Local RDD Violation	Requester's EE Context detected an invalid RDD on an outbound packet	locally detected error	RD: Class B else NA
Remote RDD Violation	Responder's Receive Queue detected a RDD violation	NAK-Invalid RD Request	RD: Class B else NA
Remote Q_Key Violation	Responder's Receive Queue detected a Q_Key violation	NAK-Invalid RD Request	RD: Class B else NA
Length error	RDMA READ response message contained too much or too little payload data.	locally detected error	RC: Class B RD: Class B else NA
Bad response	Unexpected opcode for the response packet received at the expected response PSN.[b]	locally detected error	RC: Class B RD: Class B else NA
Ghost Acknowledge	Requester received an acknowledge message at other than the expected response PSN.	locally detected error	RC: Class E RD: Class E Else NA
CQ overflow	Despite actual execution of the message, and acknowledgement, the completion notification could not be written to the CQ.	locally detected error	All: Class F

a. Local operations errors tend to be very implementation specific; not all CA's may have or detect these.
b. For example; RDMA read instead of Acknowledge, NAK code in AETH of an RDMA read, or "RDMA READ Response last" instead of middle. Note that there are specific exceptions that an implementation may elect to not treat as a bad response which are: Out of order RDMA READ Response Opcodes in the case where the requester had generated a duplicate RDMA READ Request, or the reception of an ACK instead of an RDMA READ Response of Atomic Response. This ACK may be the result of an unsolicited ACK sent by the responder that arrives at the requester before the expected RDMA READ or Atomic Response. The requester may drop this ACK packet with no ill effects.

Table 57 Summary of Requester Fault Behavior Classes

Fault Behavior Class	Current Send Queue WQE	Subsequent Send Queue WQEs	Final Send Queue State
Requester Class A	no impact	no impact	no change
Requester Class B	completed in error	flushed	error state
Requester Class C	completed in error[a]	flushed[a]	error state[a]
Requester Class D[b]	completed in error	flushed	error state
Requester Class E[c]	no impact	no impact	no change
Requester Class F	unknown	unknown	error state

a. It is possible that this class of error will render the entire HCA unable to continue work in which case the queue error states may be unknown.
b. Classes B and D are similar, however Class D applies to reliable datagram service only and also specifies that the requester's EE Context transition to the error state.
c. Classes A and E look similar, but Class A requires a retry, Class E results in no action.

9.9.2.4 REQUESTER SIDE ERROR RESPONSE

There are five different sets of error response behaviors that the requester must implement. Which behavior is executed for any given error is shown above in Table 56. This section specifies the error response behaviors.

9.9.2.4.1 REQUESTER CLASS A FAULT BEHAVIOR

Class A errors are those that are recoverable by the transport through a retry mechanism. If the retry succeeds, there is no visible impact to the transport's client (e.g. verbs layer).

The only Class A errors are Packet Sequence Error, Implied NAK sequence error, Local Ack Timeout error and an RNR NAK. Packet Sequence Error and RNR NAK are both remotely detected. A Local Ack Timeout error and an Implied NAK sequence error are detected locally by the requester.

C9-212: For an HCA using Reliable Connection service, each time the transport retries a Requester Class A error, it shall decrement a retry counter. There is one retry counter associated with Packet Sequence Errors and Local Ack Timeout errors, and a different retry counter associated with RNR NAKs. As long as the retry count has not expired the transport may continue to retry these errors. The protocol for retrying these errors is given in Section 9.7 Reliable Service on page 280.

o9-153: If a TCA requester implements Reliable Connection service, or if a CA requester implements Reliable Datagram service, each time the requester retries a Requester Class A error, it shall decrement a retry counter. There is one retry counter associated with Packet Sequence Errors and Local Ack Timeout errors, and a different retry counter associ-

ated with RNR NAKs. As long as the retry count has not expired the transport may continue to retry these errors. The protocol for retrying these errors is given in Section 9.7 Reliable Service on page 280.

C9-213: For an HCA requester using Reliable Connection service, since Requester Class A errors are recoverable, the requester shall not report them to the transport's client unless the retry count expires.

o9-154: If a TCA requester implements Reliable Connection service, or if a CA requester implements Reliable Datagram service, since Requester Class A errors are recoverable, the requester shall not report them to the transport's client unless the retry count expires.

See Section 10.10.2.2 Completion Errors on page 531 for a discussion of how errors are reported for an HCA once the retry count has expired.

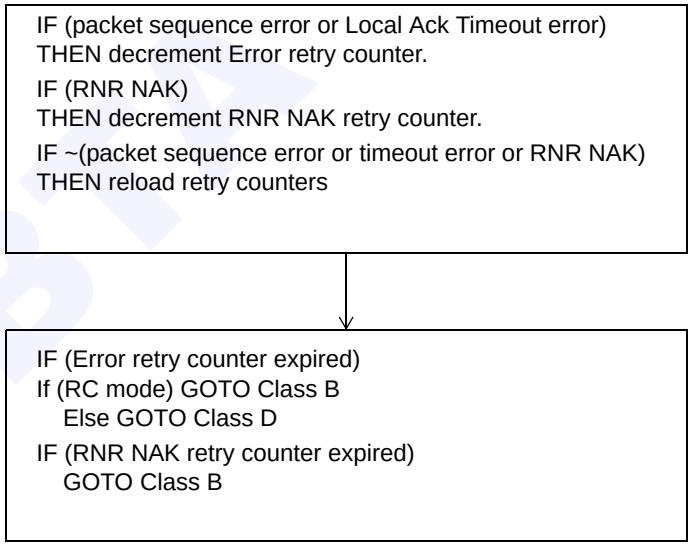


Figure 116 Requester Class A Fault Behavior

9.9.2.4.2 REQUESTER CLASS B FAULT BEHAVIOR

C9-214: In response to a Requester Class B error, for services other than Reliable Datagram, the requester shall complete the current WQE in error, transition the Send Queue to the error state and mark any subsequent WQEs posted to the Send Queue as flushed.

o9-154.a1: For CAs that implement Reliable Datagram service, the requestor, in response to a Requester Class B error, shall perform the actions described in Section 9.7.8.5 Handling QP errors - RESYNC on page 368. If, following the RESYNC process described, the message is still in error, the requester shall complete the current WQE in error, transition the Send Queue to the error state and mark any subsequent WQEs posted to the Send Queue as flushed.

For an HCA, the error is posted as “Completion - Processing type” with the appropriate error type (See 10.10.2.2 Completion Errors on page 531 for more details).

The queue shall be transitioned to the error state by the transport layer to prevent a race condition that can occur if the client (e.g. the verbs layer for an HCA) posts further WQEs to the Send Queue before it discovers that an error has occurred. This is consistent with the Send Queue state diagram as shown in Figure 124 QP/EE Context State Diagram on page 452.

Finally, all WQEs in the Send Queue behind the failed WQE are also completed with the “Completed - Flushed in Error” status.

For RC mode, note that some of these requests may have been committed to the wire by the requester, and may even have been executed and completed by the responder. It is not possible to prevent this since the responder may have executed the request before the requester detects a local error. Therefore, the responder’s local state must be considered unknown.

For reliable datagram service, the requester’s EE Context terminates the current message transfer, signals the error to the currently scheduled Send Queue, and removes the currently scheduled Send Queue from the scheduler. The EE Context then schedules the next Send Queue requesting service and proceeds. The Send Queue which caused the error behaves as described above.

C9-215: While the Send Queue is in the error state, it must silently discard any acknowledge messages that arrive.

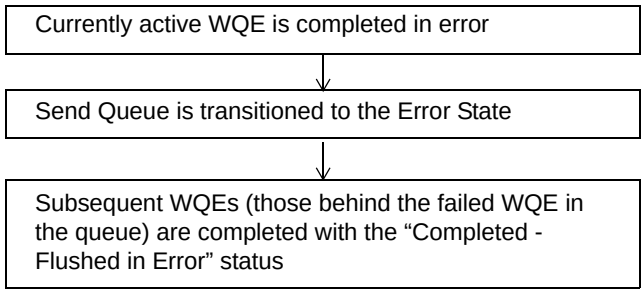


Figure 117 Requester Class B Fault Behavior

9.9.2.4.3 REQUESTER CLASS C FAULT BEHAVIOR

Since a Class C error cannot be associated with any particular WQE, it is not possible to mark a specific WQE as completed in error.

C9-216: If the Requester Class C error can be associated with a QP, the Send Queue shall be transitioned to the error state, and all uncompleted WQEs are completed with the “Completed - Flushed in Error” status.

o9-155: If a CA requester implements Reliable Datagram service, and if the Requester Class C error can be associated with an EE Context, its send side shall be transitioned to the error state, and for an HCA, the error posted is, “Affiliated Asynchronous Error”.

See 11.6.3.2 Affiliated Asynchronous Errors on page 639 for more details on HCA error reporting.

If the Requester Class C error cannot be associated with a particular resource, then the error may not be reportable in any detail or at all. See 11.6.3.4 Unaffiliated Asynchronous Errors on page 641 for HCA error handling in this case.

9.9.2.4.4 REQUESTER CLASS D FAULT BEHAVIOR

A Class D error only occurs for reliable datagram service.

o9-156: If a CA requester implements Reliable Datagram service, it shall behave as follows. For the Requester Class D error class, the transport shall transition the requester’s EE Context to the error state, terminate the current message transfer, signal the error to the currently scheduled Send Queue, and de-queue the currently scheduled Send Queue. While remaining in error state, the EE Context continues to transition to error state any other Send Queue requesting service.

Each Send Queue (QP) behaves as though for a Class B error; it marks its current WQE as completed in error, transitions the QP to the error state, and flushes all subsequent WQEs.

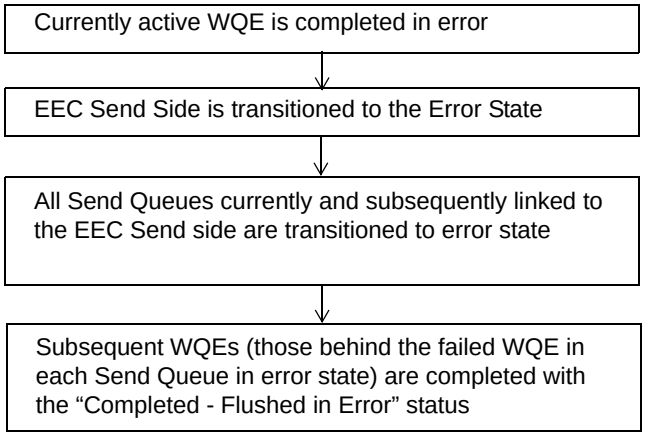


Figure 118 Requester Class D Fault Behavior

9.9.2.4.5 REQUESTER CLASS E FAULT BEHAVIOR

A Class E error occurs when the requester receives an acknowledge message with a PSN which does not match its expected PSN. These errors occur only for reliable classes of service.

These errors are not reported to upper layers.

An acknowledge message with an unexpected PSN is presumed to represent a “ghost” acknowledge message, or a duplicate acknowledge message.

A ghost acknowledge message is an acknowledge message that has been in the fabric long enough that it has survived the destruction of a connection and the subsequent establishment of a new connection.

A duplicate acknowledge message occurs when the requester, believing that its original request message is lost in the fabric, re-sends the request message. If both request messages eventually arrive at the responder, the responder may generate an acknowledge message for each of them.

C9-217: For an HCA requester using Reliable Connection service, in response to a Requester Class E error, the requester shall drop the acknowledge message. There is, however, an exception to this rule. For reliable connected service, a duplicate acknowledge message may be used by the responder to carry end-to-end flow control credits to the requester (an “unsolicited acknowledge”). Thus, if the PSN of the acknowledge message is one less than the requester’s expected PSN, the requester must recover the end-to-end credits and discard the remainder of the message. This behavior is detailed in section 9.7.7.2 End-to-End (Message Level) Flow Control on page 347.

o9-157: If a TCA requester implements Reliable Connection service, or if a CA requester implements Reliable Datagram service, in response to a Requester Class E error, the requester shall drop the acknowledge message. There is, however, an exception to this rule. For reliable connected service, a duplicate acknowledge message may be used by the responder to carry end-to-end flow control credits to the requester (an “unsolicited acknowledge”). Thus, if the PSN of the acknowledge message is one less than the requester’s expected PSN, the requester must recover the end-to-end credits and discard the remainder of the message. This behavior is detailed in 9.7.7.2 End-to-End (Message Level) Flow Control on page 347.

It should be noted that even if the Acknowledgment was an actual ghost, with wrong credits, the credit mechanism would eventually recover with no errors reported to the upper layers.

9.9.2.4.6 REQUESTER CLASS F FAULT BEHAVIOR

C9-218: A Requester Class F error occurs when the CQ is inaccessible or full and an attempt is made to complete a WQE. The Affected QP shall be moved to the error state and affiliated asynchronous errors generated as described in 11.6.3.1 Affiliated Asynchronous Events on page 637. The current WQE and any subsequent WQEs are left in an unknown state.

9.9.3 RESPONDER SIDE BEHAVIOR[13]

Table 58 lists the errors that must be detected by the responder, and the Fault Behavior Class for each error. The Fault Behavior Class specifies whether the responder returns a NAK code, whether the error is reported to the local client, and the subsequent behavior of the Receive Queue. The syndrome column lists the NAK code that is returned to the requester.

For convenience, a summary of the fault behavior classes is shown in Table 59 Summary of Responder Fault Class Behaviors on page 412.

The error detection for reliable service is described in section 9.7 Reliable Service on page 280, and error detection for unreliable service is specified in section 9.8 Unreliable Service on page 375.

C9-219: This compliance statement is obsolete and has been replaced byC9-219.1.1:.

C9-219.1.1: For the implemented subset of transport services, responders shall conform to the error behavior as specified in Table 58. Also, the responder's Receive queue shall behave as specified for each

13. For Unreliable services, a better title might be Receiver side Behavior.

Fault Behavior Class shown in Table 59 and each of the sub-sections below.

Table 58 Responder Error Behavior Summary

Error	Description	Service	Syndrome	Fault Behavior Class
Malformed WQE	Responder detected a malformed Receive Queue WQE while processing the packet.	RC, RD	NAK-Remote Operational Error	Responder Class A
		Else	NA	Responder Class A
Unsupported or Reserved OpCode	Inbound request OpCode was either reserved, or was for a function not supported by this QP. E.G. RDMA or ATOMIC on QP not set up for this. For RC this is "QP Async affiliated"	RC	NAK-Invalid Request	Responder Class C
		RD[d]		Responder Class B or Responder Class F
		else	NA	Responder Class D
Misaligned ATOMIC	VA does not point to an aligned address on an atomic operation	RC	NAK-Invalid Request	Responder Class C
		RD[d]		Responder Class B or Responder Class F
Too many RDMA READ or ATOMIC Requests	There were more requests received and not ACKed than allowed for the connection	RC	NAK-Invalid Request	Responder Class C
		RD		Responder Class B
Out of Sequence Request Packet	PSN of the inbound request is outside the responder's valid PSN window.	RC, RD	NAK-Sequence error	Responder Class B
		UC	NA	Responder Class D
Out of Sequence OpCode, current packet is "first" or "Only"	The Responder detected an error in the sequence of OpCodes; a missing "Last" packet	RC	NAK-Invalid Request	Responder Class C
		RD[d]		Responder Class B or Responder Class F
		UC	NA	Responder Class D1
Out of Sequence OpCode, current packet is not "first" or "Only"	The Responder detected an error in the sequence of OpCodes; a missing "First" packet	RC	NAK-Invalid Request	Responder Class C
		RD[d]		Responder Class B or Responder Class F
		UC	NA	Responder Class D
RESYNC Opcode incomplete WQE	The Responder has a partially complete WQE when a valid RESYNC arrives "Requestor Aborted"	RD	NA	Responder Class E
R_Key Violation	Responder detected an R_Key violation while executing an RDMA request.	RC	NAK-Remote Access Violation	Responder Class C
		RD[d]	NAK-Remote Access Violation	Responder Class B or Responder Class F
		UC	NA	Responder Class D

Table 58 Responder Error Behavior Summary (Continued)

Error	Description	Service	Syndrome	Fault Behavior Class
Local QP Error	Responder detected a local QP related error while executing the request message. The local error prevented the responder from completing the request. The local QP includes the shared receive queue, if one exists. A local QP error also occurs if a receive queue which is associated with a shared receive queue is unable to fetch a WQE from the shared receive queue due to an error condition in the shared receive queue.	RC, RD	NAK-Remote Operational Error	Responder Class A
		Else	NA	Responder Class A
Q_Key Violation	Responder's Receive Queue detected an invalid Q_Key in the request message[a]	RD[d]	NAK-Invalid RD Request	Responder Class B or Responder Class F
		UD	NA	Responder Class D
Packet Header Violation	Responder detected a header violation that requires a silent drop as described in 9.6 Packet Transport Header Validation on page 269	RC, RD UC, UD	none	Responder Class D
RDD Violation	Responder's Receive Queue detected an invalid RDD	RD	NAK-Invalid RD Request	Responder Class B
Invalid Dest QP	Dest QP does not exist or is not configured for RD service	RD	NAK-Invalid RD Request	Responder Class B

Table 58 Responder Error Behavior Summary (Continued)

Error	Description	Service	Syndrome	Fault Behavior Class
Resources Not Ready Error	A WQE or other resource is not currently available.	RC, RD	RNR-NAK[b]	Responder Class B
		UC[c], UD	none	Responder Class D
Length errors	1) Inbound “Send” request message exceeded the responder’s available buffer space: “Local Length Error” 2) RDMA WRITE request message contained too much or too little payload data compared to the DMA length advertised in the first or only packet. 3) Payload length was not consistent with the opcode: a: 0 byte <= “only” <= PMTU bytes b: (“first” or “middle”) == PMTU bytes c: 1byte <= “last” <= PMTU bytes 4) Inbound message exceeded the size supported by the CA port	RC	NAK-Invalid Request	Responder Class C
		RD		Responder Class F
		UC	NA	Responder Class D
		UD (non SRQ)	NA	Responder Class D
		UD (SRQ)	NA	Responder Class E
Invalid duplicate ATOMIC Request	A duplicate ATOMIC request packet is received, but the PSN does not match the PSN of a saved ATOMIC Request.	RC, RD	none	Responder Class D
CQ overflow	Despite actual execution of the message, and acknowledgement, the completion notification could not be written to the CQ.	All	none	Responder Class G
Local EEC Error	Responder detected a local EEC related error while executing the request message. The local error prevented the responder from completing the request.	RD	none	Responder Class H
Remote Invalidate Error	Incoming Send with Invalidate contains an invalid R_Key, or the R_Key contained in the IETH cannot be invalidated.	RC	NA	Responder Class J Fault Behavior

a. Q_Key violations require the incrementing of a counter and a potential trap as described in 10.2.5 Q_Keys on page 439
b. A TCA which does not support the generation of RNR-NAK, should not simply delay responding beyond the usual response time.
c. An RDMA Write with immediate in Unreliable Connected mode may perform the RDMA Write portion of the packet, before dropping the “immediate” portion because there is no WQE available.
d. For some RD error cases, it is desirable for the responder to have the option to complete the receive WQE in error, rather than leave the WQE on the receive queue. For those cases, the responder is permitted to use Class F responder behavior instead of Class B error behavior. Class F differs from Class B primarily in that it allows the receive WQE to be completed in error.

Table 59 Summary of Responder Fault Class Behaviors

Fault Behavior Class	NAK Codes Returned	Current Receive WQE[a]	Subsequent Receive WQEs	Final Receive Queue State
Responder Class A	For reliable services: Remote Operational Error	WQE completed in error	flushed	error state
Responder Class B	Invalid Request Invalid RD Request Remote Access Violation Sequence error RNR-NAK	no WQE consumed	no impact	no change
Responder Class C	Invalid Request	completed in error	flushed	error state
Responder Class D, D1	none	no WQE consumed	no impact	no change
Responder Class E	none	may be completed in error[b]	may be completed in error[c]	no change
Responder Class F	Invalid Request Invalid RD Request Remote Access Violation	completed in error	no impact	no change
Responder Class G	none	unknown	unknown	error state
Responder Class H[d]	none	completed in error	no impact	no change
Responder Class J	None	Completed in error	Flushed	Error State

a. A WQE is only completed if open for Sends and RDMA WRITE with Immediate data.
b. Disposition of the current receive WQE is dependent on whether the receive queue is associated with a shared receive queue or not. Please see section 9.9.3.1.6 Responder Class E Fault Behavior on page 416 for full details and associated compliance statements.
c. Disposition of subsequent receive WQEs is dependent on whether the receive queue is associated with a shared receive queue or not. Please see section 9.9.3.1.6 Responder Class E Fault Behavior on page 416 for full details and associated compliance statements.
d. This error class results in the EEC being put into the Error state.

9.9.3.1 RESPONDER SIDE ERROR RESPONSE

There are a total of eight classes of fault behavior described for the responder side. The fault behaviors are grouped according to whether or not an error is reported to the client on the responder side, whether or not the error is reported to the requester via a NAK code, and whether or not a WQE is consumed from the Receive Queue.

9.9.3.1.1 RESPONDER CLASS A FAULT BEHAVIOR

Class A errors are traceable to a poorly formed or invalid WQE, or other error associated with the receiver QP. These errors are not caused by the sender.

C9-220: For a Responder Class A error, the error shall be reported to the responder's client, the QP is placed into the error state, and, for reliable services, a "NAK-Remote Operational Error" is generated.

For Reliable Datagram service, the EEC continues operation.

If the responder is an HCA, these errors are reported to the verbs layer as a “Completion error” or Affiliated Asynchronous error”. See 10.10.2.2 Completion Errors on page 531 for a discussion of Completion errors and 10.10.2.3 Asynchronous Errors on page 531 for a discussion of Asynchronous errors.

If the responder detects a Class A error, its behavior is as follows:

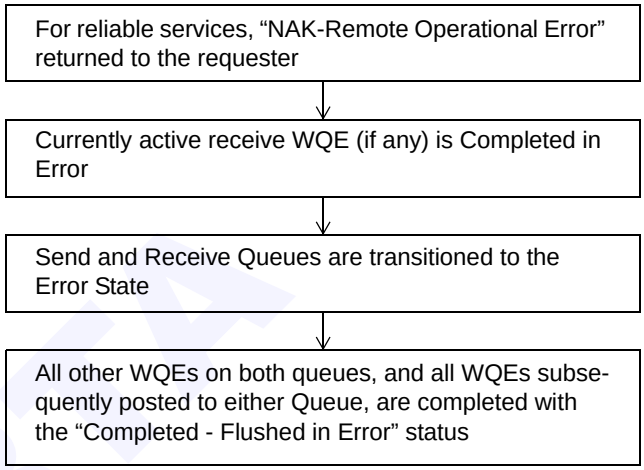


Figure 119 Transport Class A Responder Behavior

o9-157.2.1: In addition to the error behavior described above for Class A errors, if a Class A error is detected by a QP which is associated with a shared receive queue, both the WQE on which the error was originally detected and all subsequent WQEs on both the send and receive queues including WQEs subsequently fetched by the same receive queue from the associated shared receive queue, shall be marked in error. The QP on which the error was detected shall be transitioned to the error state. There shall be no state change for the shared receive queue nor is there any change in state for any other receive queues sharing the same SRQ.

o9-157.2.2: If a Class A error is detected by a shared receive queue, the error shall be reported to the responder’s client and the QP shall be placed in the error state. If the responder is an HCA, these errors are reported to the verbs layer as a “Completion error” or “Affiliated Asynchronous error”.

Note that if the error is detected by the shared receive queue (as opposed to being detected by the receive queue), a “NAK-Remote Operational Error” may not be returned since the SRQ is not required to generate acknowledgements. This is slightly different from the case where the error is

detected by the receive queue, in which case the receive queue is required to returned a “NAK-Remote Operational Error”.

o9-157.2.3: If a receive queue attempts to fetch a WQE from a shared receive queue which is in the error state, the QP shall be transitioned to the error state. If the responder is an HCA, this error is reported to the verbs layer as an “Affiliated Asynchronous error”.

9.9.3.1.2 RESPONDER CLASS B FAULT BEHAVIOR

Class B errors are reported to the requester, but are not reported to the responder’s local client.

C9-221: For an HCA requester using Reliable Connection service, and for a Responder Class B responder side error, the transport shall generate a NAK code, but shall not consume a WQE from the Receive Queue or transit the receive queue to the error state.

o9-158: If a TCA responder implements Reliable Connection service, or if a CA responder implements Reliable Datagram service, it shall behave as follows. For a Responder Class B responder side error, the transport shall generate a NAK code, but shall not consume a WQE from the Receive Queue or transit the receive queue to the error state.

Note that this fault behavior class is limited to reliable services only.

If the responder detects a Class B error, it behaves as follows:

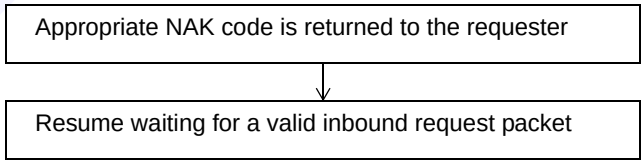


Figure 120 Responder Class B Fault Behavior

9.9.3.1.3 RESPONDER CLASS C FAULT BEHAVIOR

C9-222: This compliance statement is obsolete and has been replaced by C9-222.1.1:.

C9-222.1.1: For an HCA responder using Reliable Connection service, for a Class C responder side error, the error shall be reported to the requester by generating the appropriate NAK code as specified in Table 58 Responder Error Behavior Summary on page 409. If the error can be related to a particular QP but cannot be related to a particular WQE on that receive queue (e.g. the error occurred while executing an RDMA Write Request without immediate data), the error shall be reported to the responder’s client as an Affiliated Asynchronous error. See Section 10.10.2.3 Asynchronous Errors on page 531 for details. If the error can be

related to a particular WQE on a given receive queue, the QP shall be placed into the error state and the error shall be reported to the responder's client as a Completion error. See Section 10.10.2.2 Completion Errors on page 531.

o9-159: This compliance statement is obsolete and has been replaced by o9-159.1.1:.

o9-159.1.1: If a TCA responder implements Reliable Connection service, for a Class C responder side error, the error shall be reported to the responder's client and the QP is placed into the error state. A Class C error shall also be reported to the requester by generating the appropriate NAK code as specified in Table 58.

The Receive Queue's behavior is as follows:

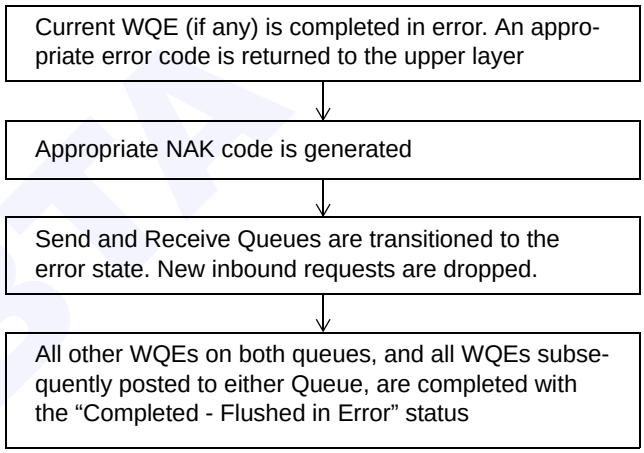


Figure 121 Transport Class C Receive Queue

See Section 10.10.2.2 Completion Errors on page 531 for more details on HCA error reporting.

9.9.3.1.4 RESPONDER CLASS D FAULT BEHAVIOR

C9-223: This compliance statement is obsolete and replaced by C9-223.1.1:.

C9-223.1.1: An inbound request packet which causes a Responder Class D error shall cause the Transport to respond as specified in 9.9.3.1.4: Responder Class D Fault Behavior.

In this case the transport shall:

- silently drop the packet (as detailed in 9.6: Packet Transport Header Validation)
- Not generate an ACK or NAK code to the requester

- Not notify the responder's client

9.9.3.1.5 RESPONDER CLASS D1 FAULT BEHAVIOR

An inbound request packet which causes a Class D1 error only occurs in Unreliable Connection mode.

C9-224: For an HCA responder using Unreliable Connection service, an inbound request packet which causes a Responder Class D1 error shall cause the Transport to respond as specified in 9.9.3.1.5: Responder Class D1 Fault Behavior.

In this case the transport shall:

- silently drop the packet
- Not notify the responder's client
- terminate the current message without consuming the current receive WQE (if any)
- wait for the first packet of a new message (which may be greater than the expected PSN.)

If the present packet, (which caused the Class D1 error) has a BTH op-code of "first" or "only"; it shall be treated as the first packet of a new message.

The Current WQE (if any) shall be reset to accept the next incoming Send or RDMA WRITE with Immediate message.

"Current message" means all the packets received since the most recently received "first" or "only" OpCode, excluding the present packet (which caused the Class D1 error).

A "new message" is denoted by a packet with a BTH opcode of "first" or "only".

o9-160: If a TCA responder implements Unreliable Connection service, it shall conform to the Class D1 HCA responder behavior described in the preceding compliance statement.

9.9.3.1.6 RESPONDER CLASS E FAULT BEHAVIOR

This fault class is intended primarily for services where a failure of a particular request packet should not impact the ability of the Receive Queue to continue receiving messages.

o9-161: If a CA implements Reliable Datagram service, then a Responder Class E error shall cause the responder to do the following:

1) Abort the current message, if it is not complete. This is done by either:
   a) Reset the WQE so that it can be reused for a future message
   b) Mark the current WQE (if any) as completed in error “Requester Aborted Error”
2) Receive the new inbound message. The Receive Queue shall continue operation without a transition to the error state.:

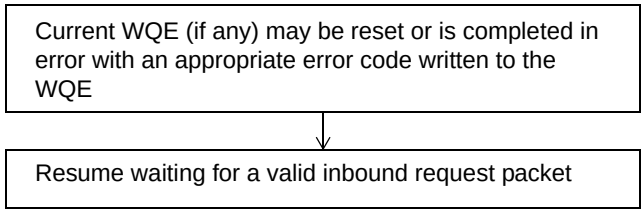


Figure 122 Transport Class E Receive Queue Behavior

Compliance statement o9-161.2.1: defines more stringent requirements which apply only to a receive queue which is associated with a shared receive queue.

o9-161.2.1: If a QP implements Unreliable Datagram service and is associated with a shared receive queue, then a Responder Class E error shall cause the responder to do the following:

1) The WQE on the receive queue that detected the error shall be completed in error.
2) Subsequent receive WQEs that had already been fetched by the same receive queue from an associated shared receive queue shall also be completed in error. Other WQEs on the shared receive queue but not fetched by the QP which detected the error remain unchanged and are not completed in error.
3) The receive queue shall continue operation without a transition to the error state.
4) The operation of the shared receive queue is not impacted by these errors.

9.9.3.1.7 RESPONDER CLASS F FAULT BEHAVIOR

Class F error behavior is for RD service only and is used to both return a NAK code to the requester and to complete a WQE in error. There are several instances where an implementation has an option to select either Class B error behavior or Class F error behavior. For these cases, the only substantial difference between Class B and Class F is in the disposition of the receive WQE which is either completed in error or left on the receive queue to be re-used. For many implementations, it is significantly simpler to complete the receive WQE in error.

o9-162: This compliance statement is obsolete and has been replaced by o9-162.1.1:

o9-162.1.1: If a CA implements Reliable Datagram service, then a Responder Class F error shall be reported both to the requester and (when a receive WQE is involved) to the responder's client. The Transport shall return the appropriate NAK code as defined in Table 58 on page 409 to the requester, and complete the current WQE (if any) in error. The Receive Queue shall continue operation without a transition to the error state.

In the case of an HCA, the error reported is a "Completion - Process Error".

Both the EEC and destination QP remain in operation.

The Receive Queue's behavior is as follows:

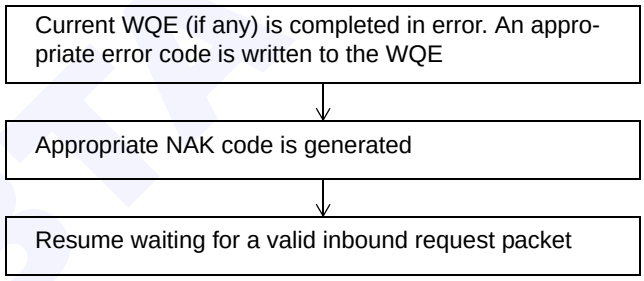


Figure 123 Transport Class F Receive Queue Behavior

9.9.3.1.8 RESPONDER CLASS G FAULT BEHAVIOR

A Class G error occurs when the CQ is inaccessible or full and an attempt is made to complete a WQE.

C9-225: A Responder Class G error occurs when the CQ is inaccessible or full and an attempt is made to complete a WQE. The Affected QP shall be moved to the error state and affiliated asynchronous errors generated as described in 11.6.3.2 Affiliated Asynchronous Errors on page 639. The current WQE and any subsequent WQEs are left in an unknown state.

9.9.3.1.9 RESPONDER CLASS H FAULT BEHAVIOR

Class H errors are traceable to a local error associated with the receiver EEC. These errors are not caused by the sender.

For a Responder Class H error, the error shall be reported to the responder's client, and the EEC is placed in the error state.

The currently active WQE should be completed in error.

If the responder is an HCA, these errors are reported to the verbs layer as either a Completion error (see Section 10.10.2.2 Completion Errors on page 531) or an Affiliated Asynchronous error (see Section 10.10.2.3 Asynchronous Errors on page 531).

9.9.3.1.10 RESPONDER CLASS J FAULT BEHAVIOR

This class of locally detected error occurs only for RC QPs. A Class J error is reported to the responder side client, but is not reported via a NAK code to the requester.

A responder class J error occurs when a responder receives a SEND with Invalidate request (either SEND last with Invalidate or SEND only with Invalidate) which contains an invalid R_Key in the IETH.

In terms of error precedence, all other errors associated with validating the packet headers and executing the SEND operation onto which the Invalidate is piggybacked are reported before an R_Key violation is reported.

o9-162.2.1: The following statements constitute the requirements for detecting and reporting an R_Key violation associated with a SEND with Invalidate request:

1) An R_Key violation, if one occurs, shall not be reported until such time as the SEND request onto which the invalidate has been piggybacked has been successfully executed. Any errors resulting from executing the SEND request must be reported before an error resulting from the invalidate operation is reported.
2) If an error occurs due to execution of the underlying SEND operation, no error related to the invalidate operation shall be reported.
3) If the underlying SEND operation executes normally, a receive WQE shall be consumed regardless of the success or failure of the associated invalidate operation. In other words, as a result of executing the underlying SEND request onto which the invalidate has been piggy-backed, a receive WQE will have been consumed. Thus, even if the invalidate operation fails, the receive WQE is always consumed.
4) The receive WQE which receives the SEND with Invalidate request shall not be completed until the corresponding invalidate operation has been completed.
5) If an error is detected in the course of executing the invalidate operation, the following actions shall occur:
    a) The WQE that experienced the SEND with Invalidate error is marked as completed in error.
    b) The QP is transitioned to the Error State.

c) Subsequent WQEs on the same RQ are marked as completed - flushed in error.

d) No NAK is sent to the Requestor.

9.10 HEADER AND DATA FIELD SOURCE

9.10.1 FIELD SOURCE WHEN GENERATING PACKETS

The following tables provide an indication of the source of the various header and data fields in the data packets for the various IBA services. The following terms are used in the table:

Link	This indicates the value is attached to the packet based on either a fixed value, or values dependent on the service, or values looked up based on parameters loaded into the logical port. Done by the link layer.
Tr	This indicates that the value is fixed or calculated by the transport layer.
QP	This indicates that the value is derived from the QP context
EE	This indicates that the value is derived from the EE context
QP+EE	This indicates that the values are derived from the QP and EE contexts
NA	Not Applicable
WQE	The value is directly or indirectly (via Address vector) derived from information in the WQE

Table 60 Packet Fields and Parameters by Service

Parameter	Description	RC	UC	RD	UD	Raw IP	Raw ET
LRH VL	The VL to use for requests. Based on SL and the port SL to VL mapping table.	link	link	link	link	link	link
LRH LVer	The version of the link level. This field depends on the revision of the device.	link	link	link	link	link	link
LRH SL	The SL to use for requests and responses	QP	QP	EE	WQE	WQE	WQE
LRH LNH IBA	IBA transport bit, indicates that BTH follows	1	1	1	1	0	0
LRH LNH GRH	GRH bit, indicates that a GRH follows	QP	QP	EE	WQE	1	0
LRH DLID	Destination local ID used for routing	QP	QP	EE	WQE	WQE	WQE
LRH Packet Length	Length of the local packet; calculated by the transport based on the message length.	WQE	WQE	WQE	WQE	WQE	WQE
LRH SLID (high bits not covered by LMC)	Source local ID in outgoing packets. From the port. With LMC low order bits (0s) added, the value is called “Base LID”.	link	link	link	link	link	link
LRH SLID (low bits covered by the LMC)	Source logical ID in outgoing packets. These LMC (as a number) bits are called the “path” bits.	QP	QP	EE	WQE	WQE	WQE
GRH IPVer”	CA’s set to 6	Tr	Tr	Tr	Tr	Tr	NA

Table 60 Packet Fields and Parameters by Service (Continued)

Parameter	Description	RC	UC	RD	UD	Raw IP	Raw ET
GRH Tclass	CA's set to 0; it will then be loaded with another value at the first encountered router. Alternately set according to application.	QP	QP	EE	WQE	WQE[a]	NA
GRH FlowLabel	CA's set to 0; it will then be loaded with another value at the first encountered router. Alternately set according to application.	QP	QP	EE	WQE	WQE[a]	NA
GRH Paylen	Length of the global packet; calculated by the transport based on the message length.	WQE	WQE	WQE	WQE	WQE[a]	NA
GRH NxtHdr	CA's set to IBA (0x1B)	Tr	Tr	Tr	Tr	WQE[a]	NA
GRH HopLmt	CA's set to 0; it will then be loaded with another value at the first encountered router. Alternately set according to application.	QP	QP	EE	WQE	WQE[a]	NA
GRH SGID	Source Global ID, from the port table and the index found in:	QP	QP	EE	WQE	WQE[a]	NA
GRH DGID	Destination Global ID	QP	QP	EE	WQE	WQE[a]	NA
BTH OpCode	Depends on operation, set by the transport layer.	Tr	Tr	Tr	Tr	NA	NA
BTH TVer	The version of the transport. This field depends on the revision of the device (0).	Tr	Tr	Tr	Tr	NA	NA
BTH P_Key	Partition Key, from the port table and the Index found in:	QP	QP	EE	QP	NA	NA
BTH DestQP	Destination QP *For RD mode responses, this is from the Request Packet Source QP as stored in the EEC	QP	QP	WQE*	WQE	NA	NA
BTH Pad	Length of packet pad; used to calculate actual data size. Calculated by the transport layer based on data size.	WQE	WQE	WQE	WQE	NA	NA
BTH SE	Solicited Event	WQE	WQE	WQE	WQE	NA	NA
BTH M	Migrate. Set by the transport dependent on the migration state.	Tr	Tr	Tr	Tr	NA	NA
BTH AckReq	Acknowledge request	Tr	0	Tr	0	NA	NA
BTH PSN	Packet Sequence Number	QP	QP	EE	QP	NA	NA
RDETH EEC	Destination EE Context	NA	NA	EE	NA	NA	NA
DETH Q_Key	Key which protects datagram QPs	NA	NA	WQE	WQE	NA	NA
DETH Source QP	Source QP. Set by transport for datagram services.	NA	NA	Tr	Tr	NA	NA
RETH	All fields of the RDMA Extended Transport Header (when used) are taken from the WQE	WQE	WQE	WQE	NA	NA	NA
AtomicETH	All fields of the ATOMIC Extended Transport Header (when used) are taken from the WQE	WQE	NA	WQE	NA	NA	NA

Table 60 Packet Fields and Parameters by Service (Continued)

Parameter	Description	RC	UC	RD	UD	Raw IP	Raw ET
AETH MSN	Message Sequence number (ACKs only)	QP	NA	EE	NA	NA	NA
AETH Syndrome	Acknowledge syndrome, computed based on operation for reliable services	QP	NA	EE+ QP	NA	NA	NA
AETH RNR-NAK timer (TTTTT)	This value is placed in the AETH.TTTTT field when sending an RNR NAK. It denotes the minimum time to wait before retrying the request.	QP	NA	QP[b]	NA	NA	NA
AETH credit count (CCCCC)	This value is placed in the AETH.CCCCC field when sending an Ack in RC mode. It denotes the number of receive WQEs available to receive Send or RDMA write with immediate messages.	QP	NA	NA	NA	NA	NA
AtomicAckETH	ATOMIC data returned; the data is loaded as defined by the R_Key and Virtual Address, stored per WQE	WQE	NA	WQE	NA	NA	NA
IETH R_Key	This is the R_KEY that the responder is being asked to invalidate in a SEND with Invalidate operation.	WQE	NA	NA	NA	NA	NA
Immediate data	Dependent on operation	WQE	WQE	WQE	WQE	NA	NA
Payload	Dependent on operation	WQE	WQE	WQE	WQE	WQE	WQE
ICRC	Calculated by transport; data dependent	link	link	link	link	NA	NA
VCRC	Calculated by Link layer; data dependent	link	link	link	link	link	link

a. Raw IP does not have a GRH, it has the similar looking IPv6 header. The Parameters are labeled GRH for convenience. This entire header is loaded from a data segment provided by the WQE.
b. The actual value of the TTTTT field depends on the reason for generating an RNR. For HCA's, when a WQE is not ready, this value comes from the QP. Otherwise, the value is determined by the implementor.

9.10.2 TRANSPORT CONNECTION PARAMETERS

The following are not sent "on the wire" but are needed to implement the protocol. This table is included to provide a better understanding of the parameters used by the transport layer to provide a connection. This list only

covers elements mentioned in the IBA specification, other elements may be needed to completely implement connections.

Table 61 Connection Parameters by Transport Service

Parameter	Description	RC	UC	RD	UD	Raw IP	Raw ET
Connect state	State of connection (Reset, RTR, RTS, Error etc.)	QP	QP	EE+ QP	QP	QP	QP
Port number	Used only if there is more than a single port	QP	QP	EE	QP	QP	QP
Global/Local header	Determines if global header is to be attached or not.	QP	QP	EE	WQE	NA	NA
MTU	Max Size of the packets allowed on this connection.	QP	QP	EE	NA	NA	NA
RNR NAK retry time	Time before performing a retry due to RNR; this is initialized by the AETH.TTTTT field in the RNR-NAK, and counts down from there.	QP	NA	QP or EE[a]	NA	NA	NA
RNR Retry init	Send Queue RNR retry count Initialization value	QP	NA	EE	NA	NA	NA
RNR Retry counter	Send Queue RNR Retry counter value	QP	NA	QP	NA	NA	NA
Local ACK Timeout	The exponent used to calculate the delay before an ACK is declared “lost”.	QP	NA	EE	NA	NA	NA
Error Retries	Send Queue retry count for sequence or time-out errors	QP	NA	EE	NA	NA	NA
MigState	Migration State (Migrated, Armed, ReArm)	QP	QP	EE	QP	NA	NA
Disable_E2E_Credits	Send queue use E2E protocol (depends on remote side’s ability to send credits)	QP	NA	NA	NA	NA	NA
Path Speed (IPD)	Controls packet emission for slower links	QP	QP	EE	WQE	WQE	WQE
PD	Protection Domain for this QP	QP	QP	QP	QP	QP	QP
RDD	Reliable Datagram Domain	NA	NA	QP+ EE	NA	NA	NA
XmitPSN	Sequence number used when sending	QP	QP	EE	QP	NA	NA
AckPSN	Sequence number expected for the ACKs	QP	NA	EE	NA	NA	NA
Rx ePSN	Sequence number expected when receiving	QP	QP	EE	NA	NA	NA
RxAckPSN	Number of unacknowledged Rx packets	QP	NA	EE	NA	NA	NA
SSN	Transmit messages Sent Sequence Number	QP	NA	NA	NA	NA	NA
Rx MSN	Message Sequence Number	QP	NA	NA	NA	NA	NA
Rx credits	Rx queue elements posted	QP	NA	NA	NA	NA	NA
LSN	Limit Sequence number (credit accounting)	QP	NA	NA	NA	NA	NA
SchQP_dequeue	QP at head of schedule queue (RD mode)	NA	NA	EE	NA	NA	NA
SchQP_enqueue	QP at tail of schedule queue (RD mode)	NA	NA	EE	NA	NA	NA

Table 61 Connection Parameters by Transport Service (Continued)

Parameter	Description	RC	UC	RD	UD	Raw IP	Raw ET
SchQP_Next	Pointer to next QP to be scheduled (RD mode)	NA	NA	QP	NA	NA	NA
Num_RDMA_Reads	Number of RDMA READs or ATOMICs supported by remote side	QP	NA	1	NA	NA	NA
RDMAR/VA/R_Key/Size or ATOMIC “result”	The “hidden” stored address(s) of RDMA READ request(s) or ATOMIC results	QP	NA	EE	NA	NA	NA
RDMA PSN# or ATOMIC PSN #	Sequence number of requested op, used to match response on a repeat, and store reply PSN	QP	NA	EE	NA	NA	NA
RDMAR/ATOMIC Use	Usage of the resource; 1=RDMAR, 0=ATOMIC	QP	NA	EE	NA	NA	NA
Rx Completion Q		QP	QP	QP	QP	QP	QP
Tx Completion Q		QP	QP	QP	QP	QP	QP
Tx WQE pointer	Points to current Send WQE and its data segments for requests	QP	QP	QP	QP	QP	QP
Tx ACK WQE pointer	Points to current Send WQE and its data segments for Completions	QP	QP	QP	QP	QP	QP
Rx WQE pointers	Points to current Receive descriptor	QP	QP	QP	QP	QP	QP

a. This value is stored with the QP or EE at the implementor’s option; depending on whether the requester implements ’suspend’ for RNR-NAK.

9.10.3 PACKET HEADER AND DATA FIELD VALIDATION

The following tables provide an indication of the validation responsibility of the various header and data fields in the data packets for the various IBA services. The following terms are used in the table:

Link	This indicates the value is checked by the link layer.
Tr	This indicates that the value is checked against fixed values or used by the transport layer to select among choices.
QP	This indicates that the value is checked against values from the QP context
EE	This indicates that the value is checked against values from the EE context
NC	The value is Not checked
NA	Not Applicable
WQE	The value is checked against information derived from the WQE

Table 62 Packet Fields Validation source by Service

Parameter	Description	RC	UC	RD	UD	Raw IP	Raw ET
LRH VL	The VL on incoming packet.	link	link	link	link	link	link
LRH LVer	The version of the link level. This field depends on the revision of the device.	link	link	link	link	link	link
LRH SL	The SL to use for requests	NC	NC	NC	NC	NC	NC
LRH LNH IBA	IBA transport bit, indicates that BTH follows	Tr(1)	Tr(1)	Tr(1)	Tr(1)	Tr(0)	Tr(0)
LRH LNH GRH	GRH bit, indicates that a GRH follows	QP	QP	EE	Tr	Tr(1)	Tr(0)
LRH DLID	Destination local ID used for routing This is always checked at the link layer against Base LID and LMC.	link QP	link QP	link EE	link	link	link
LRH Packet Length	Length of the local packet; checked against MTUCap and NeighborMTU at link, valid packet size at Transport, and data buffer size and protection values.	WQE	WQE	WQE	WQE	WQE	WQE
LRH SLID	Source local ID in ongoing packets.	QP	QP	EE	NC[a]	NC[a]	NC[a]
GRH IPVer”	Checked for the value ’6’	Tr	Tr	Tr	Tr[a]	Tr[ab]	NA
GRH Tclass	Traffic Class	NC	NC	NC	NC[a]	NC[ab]	NA
GRH FlowLabel	Flow label	NC	NC	NC	NC[a]	NC[ab]	NA
GRH Paylen	Length of the global packet; may be checked against PMTU and LRH Packet Length at link, valid packet size at Transport, and data buffer size and protection values.	WQE	WQE	WQE	WQE[a]	WQE[ab]	NA
GRH NxtHdr	Checked for the value 0x1B	Tr	Tr	Tr	Tr[a]	NC[ab]	NA
GRH HopLmt	Hop Limit	NC	NC	NC	NC[a]	NC[ab]	NA
GRH SGID	Source Global ID	QP	QP	EE	NC[a]	NC[b]	NA

Table 62 Packet Fields Validation source by Service (Continued)

Parameter	Description	RC	UC	RD	UD	Raw IP	Raw ET
GRH DGID	Destination Global ID	QP	QP	EE	NC[a]	NC[ab]	NA
BTH OpCode	Depends on operation	Tr	Tr	Tr	Tr	NA	NA
BTH TVer	The version of the transport.	Tr	Tr	Tr	Tr	NA	NA
BTH P_Key	Partition Key; checked against the port partition table[c]	QP	QP	EE	QP	NA	NA
BTH DestQP	Destination QP; checked against the valid set and QP mode by transport.	Tr	Tr	Tr	Tr	NA	NA
BTH Pad	Length of packet pad; supplements LRH Packet Length.	WQE	WQE	WQE	WQE	NA	NA
BTH SE	Solicited Event; passed to upper layers for each message	Tr	Tr	Tr	Tr	NA	NA
BTH M	Migrate. Checked and used by transport to select alternate path parameters	Tr	Tr	Tr	NC	NA	NA
BTH AckReq	Acknowledge request	Tr	NC	Tr	NC	NA	NA
BTH PSN	Packet Sequence Number	QP	QP	EE	NC	NA	NA
RDETH EEC	Destination EE Context; checked against the valid set and EE mode by transport.	NA	NA	Tr	NA	NA	NA
DETH Q_Key	Key which protects datagram QPs	NA	NA	QP	QP	NA	NA
DETH Source QP	Source QP. Passed to upper layers for each message.	NA	NA	NC	NC	NA	NA
RETH	All fields of the RDMA Extended Transport Header (when used) are validated against protection parameters associated with QP state.	QP	QP	QP	NA	NA	NA
AtomicETH	All fields of the ATOMIC Extended Transport Header (when used) are validated against protection parameters associated with QP state.	QP	NA	QP	NA	NA	NA
AETH MSN	Message Sequence number (ACKs only)	QP	NA	Tr	NA	NA	NA
AETH Syndrome	Acknowledge syndrome	Tr	NA	Tr	NA	NA	NA
AtomicAckETH	Atomic data returned; Passed to upper layers for each message.	NC	NA	NC	NA	NA	NA
IETH R_Key	This is the R_KEY that the responder is being asked to invalidate in a SEND with Invalidate operation.	[d]	NA	NA	NA	NA	NA
Immediate data	Dependent on operation; Passed to upper layers for each message.	NC	NC	NC	NC	NA	NA
Payload	Dependent on operation; Passed to upper layers for each message.	NC	NC	NC	NC	NC	NC
ICRC	Checked by transport	link	link	link	link	NA	NA
VCRC	Checked by Link layer; data dependent	link	link	link	link	link	link

a. For HCAs, this information is provided to upper layers.
b. Raw IP does not have a GRH, it has the similar looking IPv6 header. The Parameters are labeled GRH for convenience. This entire header is loaded from a data segment provided by the WQE
c. For QP1, the P_Key need only be a member of the port's Partition table, it is not checked against a QP index..
d. See details in 9.4.1.1.3 R_Key Validation for Remote Memory Invalidate on page 251

9.11 STATIC RATE CONTROL

As the traffic load increases in a fabric, resource contention increases. Congestion management is used to smooth operation, improve fabric efficiency, improve effective bandwidth, and improve average packet latency in the face of such contention.

There are a variety of mechanisms to address this problem. For this version of IBA, only static rate control will be defined. Future versions of the specification may provide definition of additional mechanisms.

Static rate control is the ability of an endnode to keep the rate of data sourcing into the fabric below a pre-configured value.

IBA supports Static Rate control in CAs to reduce congestion caused by a higher-speed CA injecting packets onto a path within a subnet at a rate that exceeds the path or destination CA's ability to sink. For example, a CA with a 10 Gbps link transmitting packets to a CA with a 2.5 Gbps link through an intermediate switch. In this case, the switch would be required to introduce "back-pressure" (limit the link-level flow control credits returned to the faster link) in order to prevent the slower link from being overrun.

IBA provides three mechanisms to manage static rate control.

- Device provided port rate information (see 16.3.3.1 ClassPortInfo on page 991)
- FM supported reporting of best possible rates for a source/destination pair (see 15.2.5.16 PathRecord on page 899).
- CA "Inter Packet delay" parameters in the connection setup MADs (described below)

9.11.1 STATIC RATE CONTROL FOR HETEROGENEOUS LINKS

A channel adapter has the ability to limit the rate of packets injected. This rate is based on the subnet-local destination port.

o9-163: If a port can support injection into the fabric at a rate greater than 2.5 Gbits/sec, this port shall provide static rate control as defined in this section.

The link rate supported is defined by the PortInfo:LinkWidthSupported and PortInfo:LinkSpeedSupported attributes. See Table 145 PortInfo on page 822 for a description of these.

o9-164: If a port is configured for injection into the fabric at a rate greater than 2.5 Gbits/sec, it shall not schedule a packet for injection into the local subnet until a programmable amount of time has passed since the last

packet was scheduled for injection from this source port to the same destination port.

The link rate configured is defined by the PortInfo:LinkWidthActive and PortInfo:LinkSpeedActive attributes. See Table 145 PortInfo on page 822 for a description of these.

In the above, destination port refers to the full DLID (i.e. base DLID plus path bits) of the destination port within the local subnet, even for globally routed packets, while source port refers to the ingress port regardless of SLID (I.e. applies to all the SLIDs associated with this port).

The time to wait before transmitting a subsequent packet is based on the time it takes to transmit the current packet.

o9-165: If a port can support injection into the fabric at a rate greater than 2.5 Gbits/sec, the time to wait between scheduling packets destined for the same DLID and originating from the same port is determined by the Inter Packet Delay (IPD). Specifically, if a packet *b* is to be sent to the same DLID and using the same source port as packet *a*, then packet *b* shall not be scheduled until a time T_s has passed since packet *a* was scheduled, where T_s is calculated as: (IPD + 1) multiplied by the time it takes to transmit the first packet. Further, the time it takes to transmit a packet is calculated as LRH:PktLen*4/L_r where L_r is the port speed as obtained from the PortInfo:LinkWidthActive and PortInfo:LinkSpeedActive attributes.

The Inter Packet Delay (IPD) value is an 8-bit integer and is interpreted as depicted in the table below. Note that all 256 possible values are legal.

Table 63 Inter Packet Delay

IPD	rate	Comment
0	100%	Suited for matched links
1	50%	
2	33%	Suited for 30 Gbps to 10 Gbps conversion
3	25%	Suited for 10 Gbps to 2.5 Gbps conversion
11	8%	Suited for a 30 Gbps to 2.5 Gbps conversion

Note also that the above table is a sample and does not include all required IPDs. See 17.2.6 Static Rate Control on page 1029 for which values of IPD CAs are required to support.

C9-226: If a CA is requested to use an unsupported value, the CA shall pick a supported value, and return that value in the appropriate MADs or verbs.

If the requested IPD value is not supported, it is recommended that the CA use the nearest supported value that is larger than the requested value.

Each connected QP (EE context for RDs) should have a programmed IPD value. UDs should include the IPD in the WQE.

The same value of IPD should be programmed for each connected QP and WQE using the same port and same DLID. If different values of IPD are programmed, the CA may use any of these values for any of this traffic.

CHAPTER 10: SOFTWARE TRANSPORT INTERFACE

10.1 OVERVIEW

This chapter describes the software transport layer of the IBA. The software transport defines the capabilities and behavior of the Channel Interface (CI), the presentation of the channel to the Verbs Consumer. This interface is implemented as a combination of the Host Channel Adapter (HCA), its associated firmware, and host software. Specification of the API used by the Verbs Consumer to access the capabilities of the CI is outside of the scope of this architecture.

A concept frequently encountered in this specification is that of Verbs Consumer. The precise meaning of the phrase varies, as a function of context, but it always means, as defined in the Glossary, the executing entity employing the capabilities of the CI to accomplish some objective. In some instances the Verb Consumer may be a OS kernel thread, in others a user-level application, and in still others, some special, privileged process. Where the difference is important to the correct behavior of an implementation, it is defined explicitly, as in 11.1 Verbs Introduction and Overview on page 546; elsewhere, it is left unspecified.

While the Partitioning section is not strictly part of the software transport layer, it describes ideas that connect intimately with the semantics of the Queue Pair (QP), and are therefore reasonably elaborated in this chapter. In addition, giving the descriptions of the necessary entities here ensures their inclusion in the architecture specification.

10.1.1 INTRODUCTION

The CI is the locus of interaction between the consumer of IBA services and the instantiation of an IBA fabric. Access to the HCA is via Verbs, which enable creation and management of QPs, management of the HCA, and coping with error indications from the CI that may be surfaced via the Verbs. All these activities must be carried out so as to enable Verbs consumers to enjoy the same level of protection and security as are guaranteed other entities supported by the host operating system.

Fundamental to CI interaction is management of HCAs, which includes arranging access to them, accessing and modifying selected of their attributes, and shutting them down. These activities are described below, and details of the corresponding Verbs layer semantics are given in the next chapter.

Entities with central importance to CI operation are QPs. They must be created, associated with protection domains, modified as required, and

destroyed to free up resources when no longer needed. In use, they provide repositories for addressing information needed by Verbs consumers, as well as protection information to guarantee the operational integrity of themselves and the host system. QPs provide for various modes of operation, depending upon the requirements of the consumer. Details of these modes, as well as the means of establishing them for a QP are described below, and corresponding Verbs semantics are detailed in the next chapter. For a graphical depiction of the QP, see Figure 11 on page 91.

As a central mode of QP operation, direct, protected access to consumer memory is critical to realizing the performance potential of the IBA. This chapter describes the semantics of memory access defined for the architecture, detailing the ideas of memory regions and windows, and their registration, access keys for local and remote access to registered memory, and the management of errors that may arise in this context.

A Work Request (WR) is an elementary object in the software transport layer, used by consumers to enqueue Work Queue Elements (WQEs) to the Send and Receive queues of a QP. The WQE is what identifies the individual events of communication over the IBA fabric. A graphical depiction of the WQE and QP can be seen in Figure 12 on page 92. The WR types, and the dynamics of their creation, use, and disposition via entries in Completion Queues (CQEs) are described in the sections to follow, as are the disposition of errors that may arise as they are used. Details of their contents are given in the next chapter.

10.2 MANAGING HCA RESOURCES

10.2.1 HCA

Verbs allow the Consumer to open an HCA, retrieve HCA attributes, modify HCA attributes that can be changed by the Consumer, and close the HCA.

Queue Pairs, Completion Queues and other resources associated with a specific HCA instance cannot be shared across multiple HCAs, even if they are managed by the same device driver software.

The intent of the architecture is to allow an implementation to pass Work Requests and Completion Status to and from a user space Consumer process to the HCA without kernel involvement.

10.2.1.1 OPENING AN HCA

The Verb used to open an HCA returns an opaque object or handle to uniquely reference each HCA so that Consumers can distinguish between HCAs in the endnode.

Opening an HCA prepares the HCA for use by the Consumer. Once opened an HCA cannot be opened again until after it has been closed.

10.2.1.2 HCA ATTRIBUTES

HCA Attributes are device characteristics. These attributes must be able to be retrieved by the Consumer. The full list of HCA Attributes are defined in 11.2.1.2 Query HCA on page 551.

Additional attributes associated with the Post-1.1 Verb Extensions defined in section 11.1.1 Verb Extensions are described in section 11.2.1.2 Query HCA on page 551.

10.2.1.3 MODIFYING HCA ATTRIBUTES

Modification of a restricted set of HCA attributes is permitted. This is primarily restricted to performance and error counter management information. Most HCA Attributes are either fixed or manipulated through the Fabric Management Interface or General Services Interface.

10.2.1.4 CLOSING AN HCA

Close restores the HCA to its initialized condition, and deallocates any resources allocated during the HCA open.

It is not the responsibility of the Channel Interface to track any resources which were not allocated by the HCA open.

10.2.2 ADDRESSING

10.2.2.1 SOURCE ADDRESSING

For global addressing, each HCA Source Port has a GID Table containing the valid GIDs for the Source Port. The GID Table is obtained via the Query HCA Verb.

C10-1: For each HCA Source Port, the CI **shall** maintain a GID Table containing the valid GIDs for the Source Port.

Each Address Vector contains a Source GID Index. The Source GID Index specifies an index into the Source GID Table. The entry referenced by the Source GID Index defines the Source GID associated with the Address Vector.

C10-2: For each GID Table, the first entry in the table **shall** contain the read-only invariant value of GID index 0.

For local addressing, IBA enables the use of multiple LIDs with each HCA port through the use of a Base LID, LMC and Path Bits.

C10-3: Each Address Vector **shall** contain specific Source Port LID Path Bits.

When the HCA constructs an SLID by taking the most significant bits from the port's Base LID and the least significant bits from an Address Vector's Path Bits, the port's LMC specifies how many bits come from the latter.

10.2.2.2 DESTINATION ADDRESSING

Addressing of destination endpoints is determined based on the Service Type of the QP:

C10-4: For Connection-oriented QPs, the destination address **shall** be stored in the QP Context, and **shall** be manipulated exclusively through the Modify Queue Pair Attributes Verbs.

o10-1: If the CI supports the RD Service, then for Reliable Datagram QPs, the destination address **shall** be stored in the EE Context, and **shall** be manipulated exclusively through the Modify EE Context Attributes Verb, and an EE Context **shall** be referenced by each individual Work Request (see 10.2.7 End-to-End Contexts).

o10-2: If the CI supports Raw Datagram Service, then for Raw QPs, the destination address **shall** be supplied via each individual Work Request.

o10-2.1.1: If the CI supports Address Handle port number checking, operations on unreliable datagram queue pairs that access an Address Handle shall be allowed only if the Queue Pair's primary physical port matches the physical port of the Address Handle. If there is a mismatch, the CI shall return either Invalid Address Handle as an immediate error or Local QP Operation Error as a completion error.

If the CI does not support Address Handle port number checking, and while processing operations on UD QPs, if the Queue Pair's primary physical port number does not match the physical port number in the Address Handle, then no error is generated because of the port number mismatch.

C10-5: For Unreliable Datagram QPs, the destination address of the node **shall** be contained in an Address Handle, and an Address Handle **shall** be referenced by each individual Work Request.

An Address Handle is a consumer-opaque object that refers to a local or global destination. Verbs are used to create, modify and destroy Address Handles. Address handles are associated with protection domains. Protection domains are described in 10.2.3 Protection Domains .

The verbs consumer should avoid modifying or destroying an Address Handle while there are outstanding WRs that reference that Address Handle. In any case the CI must process any WR, which references a destroyed or modified Address Handle, and complete it either successfully or in error (according to the normal CQE generation rules). The CA may

emit the packet only when the completion is successful, however, the destination address may be indeterminate. When the completion is in error, no packet is emitted at all.

C10-5.2.1: The CI shall process any WR that references an Address Handle which was modified or destroyed while the WR has been outstanding. The WR shall be completed either successfully or in error (normal CQE generation rules apply). A packet shall only emitted when the operation completes successfully, though the destination address may be indeterminate. If the completion is in error the CI shall not emit the packet and the reported completion return status shall be Local QP Operation Error.

C10-6: The CI **shall** support sending messages from a QP or EE addressed to the same or a different QP/EE on the same port in the sending HCA. Such messages **shall not** be transmitted through the fabric, but **shall** remain contained within that HCA.

No special addressing mechanisms are necessary to accomplish this; instead, the destination information in the source QP, EE, Address Vector, or Work Request is the same as that which any other node on the fabric would use to address the destination QP/EE.

Address Vector information can be obtained from the Subnet Administrator via the SubnAdmGet method on the PathRecord attribute. The PathRecord information is used as part of an Address Vector in an Address Handle, End to End Context or Queue Pair Context. This information is then taken from those locations as per Table 64 to create the packet. If parameters other than those obtained from exactly one applicable PathRecord are used as part of an Address Vector, this can have undesirable consequences including, but not limited to, packet discard and connection teardown.

10.2.3 PROTECTION DOMAINS

A Protection Domain (PD) is used to associate Queue Pairs (QPs) and Shared Receive Queues with Memory Regions, as a means for enabling and controlling HCA access to Host System memory.

Additionally, a Protection Domain (PD) is used to associate Queue Pairs (QPs) with Unbound Memory Windows, as a means for enabling and controlling HCA access to Host System memory.

PDs are also used to associate Queue Pairs with Bound Memory Windows as follows:

- for Type 1 Memory Windows, the PD is used to associate the Queue Pairs to a Bound Type 1 Memory Window;

- for Type 2A Memory Windows, the QP Number is used to associate the Queue Pairs to a Bound Type 2A Memory Window; and
- for Type 2B Memory Windows, the PD and QP Number are used to associate the Queue Pairs to a Bound Type 2B Memory Window.

See section 10.6.7.2.2 Remote Access Through Memory Windows on page 492 for a description of the different Memory Window Types.

PDs are also used to associate Unreliable datagram queue pairs with Address Handles, as a means of controlling access to UD destinations. Queue Pairs are described in 11.2.4 Queue Pair on page 566. Memory Regions and Memory Windows are described in detail in Section 11.2.8 Memory Management on page 592. PDs are specific to each HCA.

C10-7: Each Queue Pair in an HCA **shall** be associated with a single PD. Multiple Queue Pairs **shall** be able to be associated with the same PD.

o10-2.2.1: If the HCA supports SRQ, each Shared Receive Queue in an HCA **shall** be associated with a single PD. Multiple Shared Receive Queues and Queue Pairs **shall** be able to be associated with the same PD.

Note, the SRQ has its own PD. The SRQ may be associated with the same PD as used by one or more of its associated QPs or a different PD.

C10-8: Each Memory Region, Unbound Memory Window, or Address Handle **shall** be associated with a single PD. Multiple Memory Regions, Unbound Memory Windows, or Address Handles **shall** be able to be associated with the same PD.

C10-9: This compliance statement has been obsoleted.

C10-9.2.1: If a QP is not associated with an SRQ, operations on a Queue Pair that access a Memory Region shall be allowed only if the Queue Pair's PD matches the PD of the Memory Region.

o10-2.2.2: If the HCA supports SRQ and a QP is associated with an SRQ,

- incoming send operations that access a MR or PMR **shall** be allowed only if the SRQ's PD matches the PD of the Memory Region,
- incoming RDMA or Atomic operations that access an MR **shall** be allowed only if the QP's PD matches the PD of the Memory Region, and
- incoming RDMA, Atomic, or Invalidate operations that access a PMR **shall** be allowed only if the QP's PD matches the PD of the Memory Region.

Access to Memory Windows is dependent on the type of Memory Window selected by the Consumer when the Memory Window was Allocated through the Allocate Memory Window Verb (see section 11.2.8.9 Allocate Memory Window on page 606).

C10-10: Operations on unreliable datagram queue pairs that access an Address Handle **shall** be allowed only if the Queue Pair's PD matches the PD of the Address Handle. If there is a mismatch, the Channel Interface **shall** return either Invalid Address Handle as an immediate error or Local QP Operation Error as a completion error.

10.2.3.1 ALLOCATING A PROTECTION DOMAIN

Protection Domains are allocated through the Verbs.

A PD is required when creating a Queue Pair, registering a Memory Region, allocating a Memory Window, or creating an Address Handle.

A PD has no IB architected attributes. Operating Systems are commonly expected to enforce the policy that when a Verbs consumer creates a Queue Pair, registers a Memory Region, allocates a Memory Window, or allocates an Address Handle, it must specify a PD for association with the IB resources owned by it (that is, that were allocated by it).

10.2.3.2 DEALLOCATING A PROTECTION DOMAIN

Protection Domains are deallocated through the Verbs.

C10-11: This compliance statement has been obsoleted.

C10-11.2.1: If the CI does not support SRQ, the PD **shall not** be deallocated if it is still associated with any Queue Pair, Memory Region, Memory Window, or Address Handle. If this is attempted, the Verbs **shall** return an immediate error.

C10-11.2.2: If the CI supports SRQ, aPD **shall not** be deallocated if it is still associated with any Queue Pair, Memory Region, Memory Window, Address Handle, or Shared Receive Queue. If this is attempted, the Verbs **shall** return an immediate error.

10.2.4 QUEUE PAIRS

The Verb consumer uses a Verb to submit a Work Request (WR) to a Send queue or a Receive queue. Associated Send and Receive queues are collectively called a Queue Pair (QP); these QPs drive the channel interface. A QP, which is a component of the channel interface, is not directly accessible by the Verbs consumer and can only be manipulated through the use of Verbs. See 10.8 Work Request Processing Model for a description of the WR submission process.

10.2.4.1 CREATING A QUEUE PAIR

Queue Pairs are created through the Verbs.

When a QP is created, a complete set of initial attributes must be specified by the Consumer. The attributes that need to be defined when the QP is created are denoted in 11.2.4.1 Create Queue Pair on page 566.

The maximum number of Work Queue Entries (WQEs) the Consumer expects to be outstanding on each work queue of the Queue Pair must be specified when the QP is created. The actual number of entries is returned through the Channel Interface for each work queue.

When setting the maximum number of outstanding work requests on a work queue, the consumer must take into account that this number must be large enough to encompass the number of work requests on that queue that have not completed plus the number of completed work requests for that queue that have not been freed through the associated CQ (see 10.8.5.1 Freed Resource Count on page 520). Note for unsignaled completions, the consumer cannot consider the work request completed until the work request has been confirmed completed as per 10.8.6 Unsignaled Completions on page 521.

10.2.4.2 QUEUE PAIR ATTRIBUTES

Queue Pairs have attributes that can be retrieved through the Query Queue Pair Verb. The complete list of QP Attributes is described in 11.2.4.3 Query Queue Pair on page 576.

10.2.4.3 MODIFYING QUEUE PAIR ATTRIBUTES

Certain QP Attributes may be modified after the QP has been created. The subset of QP Attributes which can be modified are defined in 11.2.4.2 Modify Queue Pair on page 568.

It is possible to modify the QP Attributes with Work Requests outstanding on the QP. Any Work Requests outstanding on the specified QP may not execute properly when the attributes are changed.

When setting the maximum number of outstanding work requests on a work queue, the consumer must take into account that this number must be large enough to encompass the number of work requests on that queue that have not completed plus the number of completed work requests for that queue that have not been freed through the associated CQ (see 10.8.5.1 Freed Resource Count). Note for unsignaled completions, the consumer cannot consider the work request completed until the work request has been confirmed completed as per 10.8.6 Unsignaled Completions.

A CI may support the ability to modify the maximum number of outstanding Work Requests on a QP. If it does so, it must be able to support it while Work Requests are outstanding. In addition, it must support resizing both work queues on every QP. If immediate errors are returned, the work queue(s) must be in the same state as it was prior to the attempt to resize the work queue(s). It is understood that this may adversely affect performance, but it must not be the cause of immediate, completion or asynchronous errors, with the exception of immediate errors returned by the Modify Queue Pair Verb. Note that a resize operation may adversely affect other QPs attempting to communicate with the QP during the resize operation in the form of timeouts and retries. It may also result in the loss of data in the form of dropped packets for unreliable service type QPs.

10.2.4.4 DESTROYING A QUEUE PAIR

Queue Pairs are destroyed through the Channel Interface.

When a QP is destroyed, any outstanding Work Requests are no longer considered to be in the scope of the Channel Interface. It is the responsibility of the Consumer to be able to clean up any associated resources.

Destruction of a QP releases any resources allocated below the Channel Interface on behalf of the QP. Outstanding Work Requests will not complete after this Verb returns.

o10-2.2.3: If the CI supports IBA Unreliable Datagram Multicast (see 10.5 Multicast Services on page 465), and the Destroy Queue Pair Verb is invoked while the target QP is still attached to one or more multicast groups, the CI **shall** return an immediate error and the QP **shall not** be destroyed.

It is good programming practice to modify the QP into the Error state and retrieve the relevant CQEs prior to destroying the QP. Destroying a QP does not guarantee that CQEs of that QP are deallocated from the CQ upon destruction. Even if the CQEs are already on the CQ, it might not be possible to retrieve them. It is good programming practice not to make any assumption on the number of CQEs in the CQ when destroying a QP. In order to avoid CQ overflow, it is recommended that all CQEs of the destroyed QP are retrieved from the CQ associated with it before resizing the CQ, attaching a new QP to the CQ or reopening the QP, if the CQ capacity is limited.

If this QP is part of a connection, the connection should be released before the QP is destroyed. See section 12.9.4 State Diagram Notes on page 686 for further information.

Type 1 MWs are never bound to a QP. Therefore, Type 1 MWs do not affect the ability to destroy a QP.

o10-2.2.4: If an HCA supports the Base Memory Management extensions and supports Type 2A Memory Windows, then the HCA **must** return an error if the Consumer attempts to destroy a QP while Type 2A MWs are still bound to the QP.

o10-2.2.5: If an HCA supports the Base Memory Management extensions and supports Type 2B Memory Windows, then the HCA **must** allow the Consumer to destroy the QP while Type 2B MWs are still bound to the QP.

For a description of Memory Window types see section 10.6.7.2.2.

10.2.4.5 SPECIAL QPS

QPs designated as special are the SMI QP (QP0), GSI QP (QP1) and the Raw IPv6 and Raw Ethertype QPs. These QP types are special because they have different and more restrictive properties than QPs designed for more general use.

Incoming messages to the SMI or GSI QPs may be consumed below the Verbs by a subnet management agent (SMA) or general services agent (GSA), respectively. If a MAD is consumed below the Verbs, the semantics must be consistent from the Verbs Consumer's point of view.

C10-12: Any message processing performed below the Verbs, e.g., by a SMA, **must not** disturb any WQEs and CQEs posted on behalf of the Verbs Consumer.

C10-13: The CI **shall** allow direct access to the SMI and GSI QPs only by privileged mode Consumers.

SMI and GSI QPs can share a completion queue, but neither can share one with any QP that is not of the SMI or GSI type.

Multiple Raw IPv6 and Raw Ethertype QPs are allowed on a single HCA. However, the demultiplexing algorithm that is applied to receiving messages between QPs is outside the scope of this specification.

10.2.5 Q_KEYS

A Queue Key (Q_Key) is a construct used in Datagram Service QPs to validate a remote sender's right to access a local Receive Queue. They are set through the Modify Queue Pair Verb, as well as within Work Requests. The values used for Q_Keys are not managed below the Verbs. Q_Keys are contained in the headers for IB Datagram packets.

Q_Keys have the following properties:

- A Q_Key must be established in the QP Context before Receive Descriptors can be posted to a QP.

- The Q_Key is a modifier in the Post Send Request Verb.

C10-14: For UD Service type QPs, except QP0 and QP1, the Q_Key in the QP Context **shall** be used to validate incoming packets. If the Q_Key does not match, the packet **shall** be silently dropped. See Section 9.6.1.4.1 for rules on validating the Q_Key in incoming packets on QP0 and QP1.

o10-3: If the CI supports the RD Service, then for RD QPs, the Q_Key in the QP Context **shall** be used to validate incoming packets. If the Q_Key does not match, the packet **shall** be NAK'd. This NAK **shall** result in the Send Queue at the remote node being placed into the appropriate error state as per the state diagram.

C10-15: A Q_Key **shall** be a modifier in the Post Send Request Verb for Datagram Service Type queue pairs. The Channel Interface **shall** examine the Q_Key in the Work Request. If the high order bit of the Q_Key is set, the outgoing packet **shall** contain the Q_Key from the QP Context. If the high order bit of the Q_Key is not set, the outgoing packet **shall** contain the Q_Key from the Work Request.

For the RD service class, Q_Key violation results in the source Send Queue transitioning to the error state. The destination has the option to support a Q_Key violation counter and trap. This optional counter may be queried and set through the Verbs.

Q_Keys are not enforced on Raw QPs.

10.2.6 COMPLETION QUEUES

A CQ can be used to multiplex work completions from multiple work queues across queue pairs on the same HCA.

C10-16: The CI **shall** support Completion Queues (CQ) as the notification mechanism for Work Request completions. A CQ **shall** have zero or more work queue associations. Any CQ **shall** be able to service send queues, receive queues, or both. Work queues from multiple QPs **shall** be able to be associated with a single CQ.

10.2.6.1 CREATING A COMPLETION QUEUE

Completion Queues are created through the Channel Interface.

The maximum number of Completion Queue Entries (CQEs) that may be outstanding on a CQ must be specified when the CQ is created; this is known as the CQ's capacity. The actual capacity is returned through the Channel Interface. If the number of CQEs outstanding on a CQ is equal to its capacity, and another entry is added, the CQ overflows. It is the responsibility of the Consumer to ensure that the capacity chosen is sufficient for

the Consumer's operations; it must, in any case, arrange to handle an error resulting from CQ overflow.

C10-17: Overflow of the CQ **shall** be detected and reported by the CI before the next WC is retrieved from that CQ. This error **must** be reported as an affiliated asynchronous error -- see 11.6.3.2 Affiliated Asynchronous Errors on page 639.

10.2.6.2 COMPLETION QUEUE ATTRIBUTES

The only Completion Queue attribute is the capacity of the CQ. This attribute can be retrieved through the Query Completion Queue Verb.

Note that no Verb is provided which retrieves a CQ's set of associated Work Queues; the consumer is responsible for keeping track of this information, if needed.

10.2.6.3 MODIFYING COMPLETION QUEUE ATTRIBUTES

It must be possible to change the capacity of the CQ through the Channel Interface while Work Requests are outstanding on the queues associated with the specified CQ. It is understood that this may adversely affect performance, but it must not be the cause of immediate or completion errors, with the exception of immediate errors returned by the Resize Completion Queue Verb.

10.2.6.4 DESTROYING A COMPLETION QUEUE

Completion Queues are destroyed through the Channel Interface.

C10-18: If the Verb to destroy a CQ is invoked while Work Queues are still associated with the CQ, the CI **shall** return an error and the CQ **shall** not be destroyed.

Destruction of a CQ releases any resources allocated below the Channel Interface on behalf of the CQ.

10.2.7 END-TO-END CONTEXTS

An End-to-End Context (EE Context) is a local HCA resource, used by the local HCA to track messages transferred between itself and another HCA, to support Reliable Datagram Service QPs. EE Contexts are established in an HCA by the Consumer through the Verbs.

o10-4: If the CI supports RD Service, the CI **shall** support an EE Context for use by the Consumer to provide the connection between two HCA ports. Each local EE Context **shall** be connected to exactly one destination EE Context.

The Consumer must establish a communication channel between a pair of EE Contexts, one on each HCA, before RD messaging can begin between the two HCAs. This communication channel must be established using the normal connection style semantics described in Chapter 12, Communication Services.

o10-5: If the CI supports RD Service, multiple connected EE Contexts (RD channels) **shall** be allowed between HCA port pairs. These EE Contexts are allowed to have either the same or different sets of attributes.

Any Work Requests outstanding on the specified EE Context may not execute properly when the attributes are changed.

The Consumer submits RD Work Requests to an RD type QP's Send Queue. The Work Request specifies the EE Context to use to perform the actual message transfer. Work Requests may be submitted to a single RD QP that specify different EE Contexts as long as the EE Context specified is in the same RDD as the RD QP.

It is the responsibility of the Consumer to create, modify and destroy the EE Context, to use the Communication Services to gather the information necessary to transition the EE Context through the states as well as to fill out the necessary attributes for use.

It is important to note that Verbs manipulating EE Contexts, such as Create EE Context and Modify EE Context, use an EE Context handle, but Communication Management and submission of Work Requests to the Send Queue require the EE Context number. This number can be retrieved through the Query EE Context Verb.

10.2.7.1 CREATING AN EE CONTEXT

EE Contexts are created through the Channel Interface.

When an EE Context is created, a complete set of initial attributes must be specified by the Consumer. The attributes that need to be defined when the EE Context is created are denoted in Section 11.2.7.1 Create EE Context on page 584.

10.2.7.2 EE CONTEXT ATTRIBUTES

EE Contexts have attributes that can be retrieved through the Query EE Context Verb.

The complete list of EE Context Attributes is described in Section 11.2.7.3 Query EE Context on page 590.

10.2.7.3 MODIFYING EE CONTEXT ATTRIBUTES

Certain EE Context Attributes may be modified after the EE Context has been created. The subset of EE Context Attributes which can be modified are defined in Section 11.2.7.2 Modify EE Context Attributes on page 585.

It is possible to modify the EE Context Attributes when Work Requests requiring the EE Context are outstanding. Any outstanding WR which requires the specified EE context may not execute properly when the attributes are changed.

10.2.7.4 DESTROYING AN EE CONTEXT

EE Contexts are destroyed through the Channel Interface.

When an EE Context is destroyed, any outstanding Work Requests which depend on the EE Context are expected to complete with an appropriate error.

Destruction of an EE Context releases any resources allocated below the Channel Interface on behalf of the EE Context.

If this EEC is part of a Reliable Datagram Channel, the RDC should be released before the EEC is destroyed. See section 12.9.4 State Diagram Notes on page 686 for further information.

10.2.8 RELIABLE DATAGRAM DOMAINS

A Reliable Datagram Domain (RDD) is a means to associate Queue Pairs with EE contexts.

o10-6: If the CI supports RD Service, each RD QP **shall** be associated with only one RDD. Multiple RD QPs **shall** be able to be associated with the same RDD.

o10-7: If the CI supports RD Service, each EE context **shall** be associated with only one RDD. Multiple EE contexts **shall** be able to be associated with the same RDD.

o10-8: If the CI supports RD Service, WRs which specify an EE Context on an RD Queue Pair **shall** be allowed only if the RDD in the EE Context matches the RDD in the QP. If the RDDs do not match, the initiator's work request will complete with a Local RDD Violation Error, with no effect on the destination's receive queue (see 11.6.2 Completion Return Status on page 634 for the correct error code).

o10-9: If the CI supports RD Service, the CI **shall** support at least two RDDs.

The purpose of defining the RDD construct is to ensure that it is possible reliably to separate user and kernel I/O RD traffic through an HCA. Note also that realizing that separation requires two EE contexts, as well.

10.2.8.1 ALLOCATING A RELIABLE DATAGRAM DOMAIN

Reliable datagram domains are allocated through the Channel Interface, using a privileged operation.

10.2.8.2 DEALLOCATING A RELIABLE DATAGRAM DOMAIN

Reliable datagram domains are deallocated through the Channel Interface.

o10-10: If the CI supports RD Service, an RDD **shall not** be deallocated if it is still associated with any Queue Pair or EE Context. If this is attempted, the CI **shall** return an immediate error.

10.2.9 SHARED RECEIVE QUEUE

The semantics defined in this section are applicable if the Consumer associates a QP to a Shared Receive Queue.

10.2.9.1 MOTIVATION FOR SUPPORTING SRQ

Without SRQ, an RC or UD Consumer must post the number of receive WRs necessary to handle incoming receives on a given QP. If the Consumer cannot predict the incoming rate on a given QP, because, for example, the connection has a bursty nature, the Consumer must either: post a sufficient number of RQ WRs to handle the highest incoming rate for each connection, or, for RC, let message flow control cause the remote sender to back off until local Consumer posts more WRs.

Either approach is inefficient:

- Posting sufficient WRs on each QP to hold the possible incoming rate, wastes WQEs, and the associated Data Segments, when the Receive Queue is inactive. Furthermore, the HCA doesn't provide a way of reclaiming these WQEs for use on other connections.
- Letting the RC message flow control cause the remote sender to back off can add unnecessary latencies, specially if the local Consumer is unaware that the RQ is starving.

A Shared Receive Queue solves this problem by allowing multiple Queue Pairs to share Receive Work Requests and, more importantly, the Data Segments associated with Receive Work Requests. When an incoming Receive Message arrives on any QP that is associated with an SRQ, the HCA uses the next available SRQ WQE to receive the incoming data. The HCA returns Work Completions through the Completion Queue that is associated with the QP that received the incoming Send operation.

RC or UD QPs can be associated with an SRQ.

10.2.9.2 SHARED RECEIVE QUEUE CREATION

The Consumer creates an SRQ through the Create SRQ Verb.

When the SRQ is created through the Create SRQ verb:

- The Consumer requests the maximum number of WQEs that can be outstanding on the SRQ and the maximum number of scatter/gather elements per WQE on the SRQ. The CI returns the actual maximum number of WQEs that can be outstanding on the SRQ and the actual number of scatter/gather elements per WQE on the SRQ. Each must be greater than or equal to the number requested.

The full list Create SRQ attributes is described in section 11.2.3.1 Create Shared Receive Queue on page 563.

10.2.9.3 SHARED RECEIVE QUEUE MODIFICATION

The Consumer sets the SRQ Limit through the Modify SRQ verb. The SRQ Limit is armed when the Consumer sets the SRQ Limit to any value greater than zero. An SRQ Limit Reached Affiliated Asynchronous Event is generated whenever the number of SRQ WQEs is less than the SRQ Limit. The Consumer can use the SRQ Limit Reached Affiliated Asynchronous Event as an indicator that more WQEs need to be posted. The CI shall reset the SRQ Limit to zero when the SRQ Limit Reached Affiliated Asynchronous Event has been generated.

Note: A user mode Consumer can use the following sequence to reduce the number of user to privilege mode transitions upon SRQ and QP initialization:

- User mode Consumer invokes Create SRQ. This step requires a user-privileged-user mode transition. When the Create SRQ returns the Consumer is given an SRQ Handle.
- The user mode Consumer uses the SRQ Handle as an input modifier to the Post Receive verb, and posts Work Requests to the SRQ. The Consumer should post sufficient WRs, such that the total number of WRs is at least greater than the SRQ Limit the Consumer will set for SRQ.
- User mode Consumer invokes a combined Create QP and Modify SRQ, passing the SRQ Handle in the process. This step requires a user-privileged-user mode transition. A privileged mode Consumer turns the Create QP and Modify SRQ as two separate verb calls. The Create QP associates the newly create QP to the SRQ referenced by the SRQ Handle. The Modify SRQ sets the SRQ Limit for the SRQ referenced by the SRQ Handle.

Note, if the Consumer sets the SRQ Limit greater than zero on a Modify SRQ, the CI will re-arm the SRQ with the new SRQ Limit. The SRQ may be armed more than once through the Modify SRQ verb, even while it is already armed. The CI maintains one SRQ Limit value per SRQ and re-arming will overwrite this value.

Note: If the SRQ Limit is armed more than once, the Modify SRQ may return stating the SRQ Limit is armed, when in fact it is not and 1 or 2 Limit Reached Affiliated Asynchronous Events may surface. If the Consumer arms the SRQ more than once and then receives an event, then there is an ambiguity regarding which arming triggered the event. Therefore the Consumer should Query the SRQ to determine if the SRQ is still armed.

If the HCA supports the modification of the maximum number of outstanding SRQ WRs, then Consumer may request to change the maximum number of outstanding WRs through the Modify SRQ verb. The CI returns the actual maximum number of WQEs that can be outstanding on the SRQ, this value must be greater than or equal to the number requested.

o10-10.2.1: If the HCA supports modifying the maximum number of outstanding SRQ WRs, then for each SRQ supported by the HCA, the HCA shall allow the maximum number of outstanding SRQ WRs to be changed, even if WRs are still outstanding on the SRQ.

Similar to the QP resize operation, an SRQ resize may adversely affect HCA performance.

o10-10.2.2: If the HCA supports modifying the maximum number of outstanding SRQ WRs and the Consumer performs a resize operation through the Modify SRQ verb, then the HCA shall not return an error as a direct result of performing the Modify SRQ verb. The only exceptions are:

- when the SRQ size is decreased below the current number of outstanding WQEs on the SRQ, in this case, the CI must return an immediate error and not affect the SRQ state; and
- when the HCA supports modifying the maximum number of outstanding SRQ WRs and the Consumer performs a resize operation through the Modify SRQ verb that decreases the SRQ size below the SRQ Limit, the CI must return an immediate error and not affect the SRQ state.

When the local HCA is performing a resize operation, remote QPs that target local QPs associated with the SRQ being resized may experience message time-outs. These message timeouts may indirectly lead to connection teardown. Therefore, it is recommended that the Consumer either use a Consumer level message quiesce or a QP level Quiesce (i.e. through SQD) before resizing an SRQ.

10.2.9.4 SHARED RECEIVE QUEUE DESTRUCTION

The Consumer destroys an SRQ through the Destroy SRQ Verb. The attributes of the Destroy SRQ Verb are described in 11.2.3.4 Destroy Shared Receive Queue on page 565.

o10-10.2.3: If the HCA supports SRQ and the Consumer invokes the Destroy SRQ verb while there are QPs still associated with the SRQ, the CI shall return an Immediate Error.

o10-10.2.4: If the HCA supports SRQ and the Consumer invokes the Destroy SRQ verb while there are WRs still outstanding on the SRQ, the HCA shall perform the destroy SRQ operation.

o10-10.2.5: If the HCA supports SRQ, when the CI successfully returns from the Destroy SRQ verb, the CI shall free all resources associated with the SRQ, including any WRs that were posted to the SRQ, but not completed.

10.2.9.5 SRQ STATES

The SRQ only has two states: Non-Error State and Error State.

In the Non-Error State: the Consumer is allowed to post Work Requests to the SRQ; and QPs are allowed to retrieve Receive Queue WQEs from the SRQ. In the Error State: the Consumer may be able to post Work Requests to the SRQ; but QPs cannot retrieve Receive Queue WQEs from the SRQ.

When an SRQ is created it is placed in the Non-Error State. If an SRQ Catastrophic Error occurs, the SRQ is placed in the Error State. Attempting to Query or Modify an SRQ in the Error State shall return an Immediate Error. The only way for the SRQ to exit the Error State is through a Destroy SRQ. Upon successful completion of a Destroy SRQ, the SRQ exits the SRQ State logic.

10.2.10 INFINIBAND HEADER DATA AND SOURCES

The following table lists each of the data items present in an IBA protocol header, as well as some internal state needed to send packets. For each Transport Service Type, it lists the source of that data or state. The sources for each of these are accessed through the Verbs. This table is provided to establish the correlation between the packet fields and the

constructs established through the Verbs. Note that the legend for Table 64 is Table 65, below.

Table 64 Packet Fields, Queue Parameters, and their Sources

Header	Field	RC	UC	RD	UD	Raw IP	Raw ET
LRH	Virtual lane	Computed from SL and CAP SL->VL table					
LRH	LRH version	Fixed					
LRH	Service level	QP		EE	AV	WR	
LRH	LRH next header – IBA transport bit	Fixed=1				Fixed=0	
LRH	LRH next header – GRH bit	QP		EE	AV	Fixed=1	Fixed=0
LRH	Destination local identifier	QP		EE	AV	WR	
LRH	Packet length	Computed from data/header length					
LRH	Source local identifier (part not covered by LMC)	CAP					
LRH	Source local identifier (part covered by LMC)	QP		EE	AV	WR	
LRH	Reserved	Fixed=0					
GRH	IP version	Fixed=6				N/A	
GRH	Traffic class	QP		EE	AV	N/A	
GRH	Flow label	QP		EE	AV	N/A	
GRH	Payload length	Computed from data/header length				N/A	
GRH	Next header	Fixed				N/A	
GRH	Hop limit	QP		EE	AV	N/A	
GRH	Source GID	Computed from CAP table and index in QP		Computed from CAP table and index in EE	Computed from CAP table and index in AV	N/A	
GRH	Destination GID	QP		EE	AV	N/A	
BTH	OpCode	WR				N/A	
BTH	BTH version	Fixed=0				N/A	
BTH	Partition key	QP		EE	QP	N/A	
BTH	Destination queue pair	QP		WR		N/A	

Table 64 Packet Fields, Queue Parameters, and their Sources (Continued)

Header	Field	RC	UC	RD	UD	Raw IP	Raw ET
BTH	Pad count	Computed from data & header length				N/A	
BTH	Solicited event	WR				N/A	
BTH	Packet sequence number	Computed from QP state		Computed from EE state	Computed from QP state	N/A	
BTH	Reserved	Fixed=0				N/A	
RDETH	Remote EE context	N/A		EE	N/A		
RDETH	Reserved	N/A		Fixed=0	N/A		
DETH	Queue key	N/A		WR or QP depending on WR		N/A	
DETH	Source queue pair	N/A		QP		N/A	
DETH	Reserved	Fixed=0				N/A	
RETH	Virtual address	WR			N/A		
RETH	R_Key	WR			N/A		
RETH	DMA length	WR			N/A		
AtomicETH	Virtual address	WR	N/A	WR	N/A		
AtomicETH	R_Key	WR	N/A	WR	N/A		
AtomicETH	Swap data	WR	N/A	WR	N/A		
AtomicETH	Compare data	WR	N/A	WR	N/A		
IETH	R_Key	WR	N/A				
AETH	Message sequence number	Computed	N/A	Computed	N/A		
AETH	Syndrome	Computed	N/A	Computed	N/A		
RWH	Reserved	N/A					Fixed
RWH	EtherType	N/A					WR
AtomicAckETH	Original remote data	Memory	N/A	Memory	N/A		
ImmDT	Immediate Data	WR				N/A	
	Local EE context	N/A		WR	N/A		
	Port number	QP		EE	QP		
	Transport Timeout	QP	N/A	EE	N/A		

Table 64 Packet Fields, Queue Parameters, and their Sources (Continued)

Header	Field	RC	UC	RD	UD	Raw IP	Raw ET
	Retry count	QP	N/A	EE	N/A		
	RNR retry count	QP	N/A	EE	N/A		
	PMTU	QP		EE	N/A		
	Maximum static rate	QP		EE	AV	WR	
	Protection domain	QP					
	Reliable datagram domain	N/A		QP/EE	N/A		
	Send PSN	QP		EE	N/A		
	Receive PSN	QP		EE	N/A		
	Outstanding atomics/RDMA reads supported at destination	QP	N/A	EE	N/A		
	Send CQ	QP					
	Receive CQ	QP					

The following table is the legend for the previous one.

Table 65 Legend for Table 64

Abbreviation	Meaning
AETH	Acknowledgement extended transport header
AtomicAck-ETH	Atomic acknowledgement extended transport header
AtomicETH	Atomic extended transport header
AV	Part of the address vector object defined by the Verbs
BTH	Base transport header
CA	Property of the channel adapter
CAP	Property of the channel adapter port
Computed...	Calculated from other values as specified
DETH	Datagram extended transport header
DS	Field taken from data segment pointed to by work request
EE	Taken from the EE context (RD service only)
Fixed	Value is determined by the specification and is the same in all packets.
GRH	Global routing header

Table 65 Legend for Table 64 (Continued)

Abbreviation	Meaning
IETH	Invalidate Extended Transport Header
ImmDT	Immediate Data Extended Transport Header
LRH	Local routing header
Memory	Retrieved from host memory by CA
N/A	Not applicable to this Service Type
PMTU	Path maximum transfer unit
QP	Taken from Queue Pair state (the real QP in the case of RD service)
Raw	Raw Packet service
RC	Reliable Connected service
RD	Reliable Datagram service
RDETH	Reliable datagram extended transport header
RNR	Receiver not ready
RWH	Raw ethertype header
UC	Unreliable Connected service
UD	Unreliable Datagram service
WR	Taken from a Work Request

10.3 RESOURCE STATES

10.3.1 QUEUE PAIR AND EE CONTEXT STATES

This section contains the QP and EE Context state diagram. The same state diagram is used for both QPs and EE Contexts. This section will use the term QP/EE for this and, where differences are important, will note them. This section will contain a definition of the QP/EE states. The QP/EE states defined here and the transition order between the states are shown in Figure 124. EE Contexts are created only for the Reliable Datagram Service Type, whereas QPs are used for all Transport Service Types.

Note that while QPs and EE Contexts share the same state diagram, the EE Context state has no relationship to the states of the sending and receiving RD QPs using that EE Context. The reader should not assume that because a QP made a state transition that a EE Context associated with that RD QP will also transition, and vice versa.

Even though a subset of the states could occur in any order for some of the Transport Service Types, the states must transition in the order spec-

ified. This is to keep the state definitions consistent and error semantics simplified. The order chosen is that required to support the Reliable Connection Service Type, and to provide for completeness of the information needed to transfer data using an EE Context.

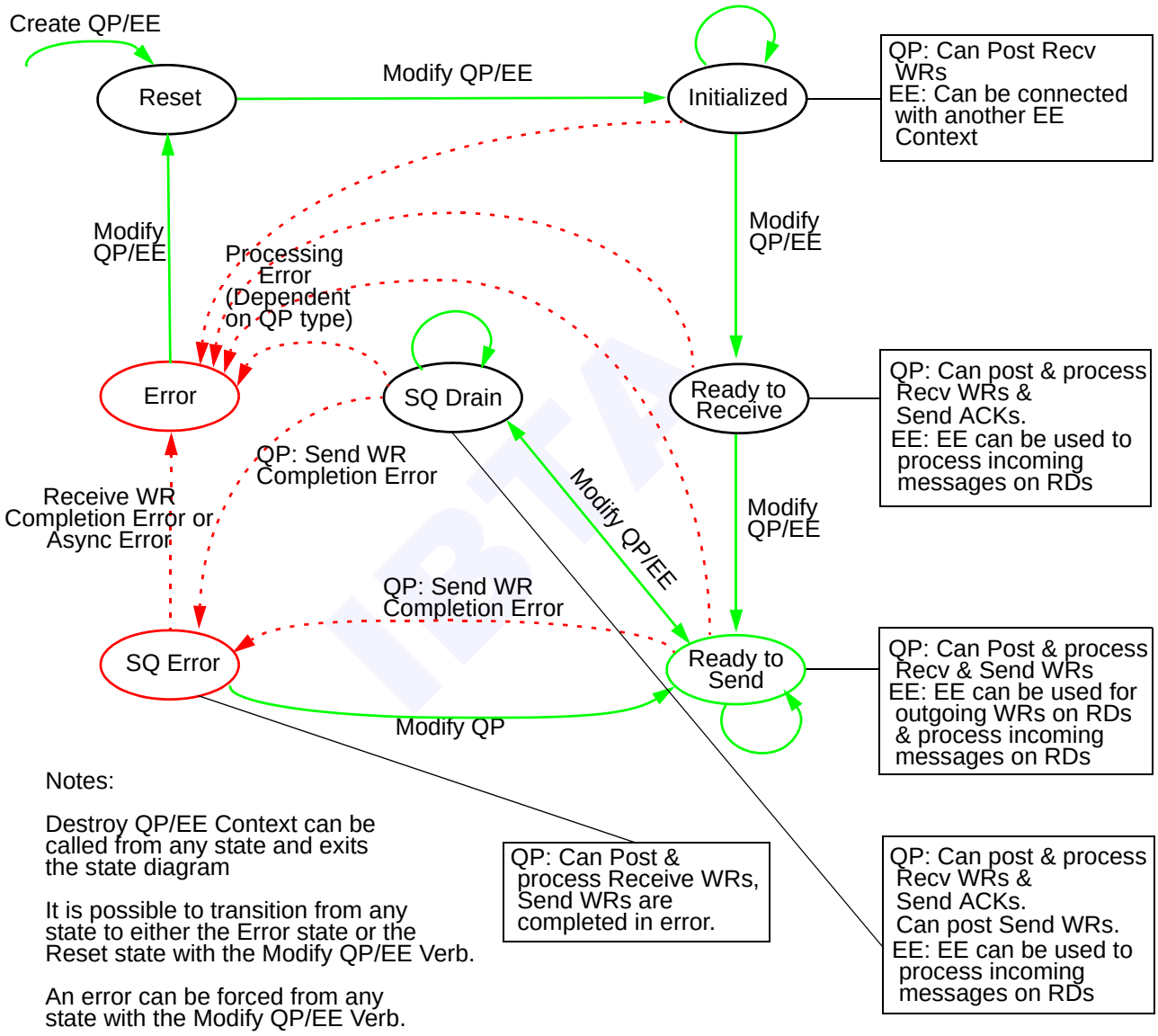


Figure 124 QP/EE Context State Diagram

Note, for QPs that are associated with an SRQ, the Consumer should take the QP through the Error State before invoking a Destroy QP or a Modify QP to the Reset State. The Consumer may invoke the Destroy QP without first performing a Modify QP to the Error State and waiting for the Affiliated

Asynchronous Last WQE Reached Event. However, if the Consumer does not wait for the Affiliated Asynchronous Last WQE Reached Event, then WQE and Data Segment leakage may occur. Therefore, it is good programming practice to tear down a QP that is associated with an SRQ by using the following process:

- Put the QP in the Error State;
- wait for the Affiliated Asynchronous Last WQE Reached Event;
- either:
  - drain the CQ by invoking the Poll CQ verb and either wait for CQ to be empty or the number of Poll CQ operations has exceeded CQ capacity size; or
  - post another WR that completes on the same CQ and wait for this WR to return as a WC;
- and then invoke a Destroy QP or Reset QP.

o10-10.2.6: If an HCA supports the Base Memory Management extensions and supports Type 2A Memory Windows, then the HCA must return an error if the Consumer attempts a Modify QP transition from any State to the Reset State while Type 2A MWs are still bound to the QP.

o10-10.2.7: If an HCA supports the Base Memory Management extensions and supports Type 2B Memory Windows, then the HCA must allow the Consumer to transition the QP from the any State to the Reset State while Type 2B MWs are still bound to the QP.

It is important to understand that the QP/EE states are intended to be used as an integral part of the connection process. A thorough understanding of the connection process and the connection state diagram is assumed. This is discussed in detail in 12.9.7.1 Active States on page 689 and 12.9.7.2 Passive States on page 691.

C10-19: With one exception, the CI **shall** implement the Reset, Init, RTR, RTS, SQD, SQEr, and Error states for each QP. The CI **shall not** implement the SQEr state for RC QPs. Transitions between those states **must** be restricted to those shown in Figure 124.

o10-11: If the CI supports RD Service, the CI **shall** implement the Reset, Init, RTR, RTS, SQD, and Error states for each EE Context. Transitions between those states **must** be restricted to those shown in Figure 124.

10.3.1.1 RESET

The characteristics of the Reset state are:

C10-20: A newly created QP/EE **shall** be placed in the Reset state.

- It is possible to transition to the Reset state from any state by specifying the Reset state when modifying the QP/EE attributes.
- Any resources required to implement the QP/EE have been allocated. For example, some implementations require WQEs and/or control structures to be allocated.
- The Modify Queue Pair and Modify EE Context Attributes Verbs are the only way for the Verbs Consumer to cause a transition out of the Reset State, without destroying the EE/QP.

C10-21: Upon creation, or transition to the Reset state, all QP/EE attributes **must** be set to the initialization defaults, as documented in the Create Queue Pair and Create EE Context Verbs.

- Transition out of the Reset state can be effected by calling the Destroy Queue Pair or Destroy EE Context Verbs, thus exiting the state diagram.

For EEs:

- It is an error for a Work Request to specify an EE Context in the Reset state.

o10-12: If the CI supports RD Service, and a Work Request is submitted to the Send Queue of an RD QP specifying an EE Context in the Reset state, the Work Request **shall** be completed in error.

- No work requests can be outstanding which use an EE Context in this state.

o10-13: If the CI supports RD Service, any incoming messages which target an EE in the Reset state **must** be silently dropped.

For QPs:

C10-22: If a Work Request is submitted to a Work Queue while its corresponding QP is in the Reset State, an immediate error **shall** be returned.

- The Work Queues are empty. No Work Requests are outstanding on the work queues.
- All Work Queue processing is disabled

C10-23: Incoming messages which target a QP in the Reset state **must** be silently dropped.

10.3.1.2 INITIALIZED (INIT)

The characteristics for the Initialized state are:

- The basic QP/EE attributes have been configured as defined in Modify Queue Pair and Modify EE Context Attributes Verbs.

- Transition into this state is only possible from the Reset state.
- The Modify Queue Pair or Modify EE Context Attributes Verbs are the only way for the Verbs Consumer to cause a transition out of the Init state, without destroying the EE/QP.
- Transition out of the Init state can be effected by calling the Destroy Queue Pair or Destroy EE Context Verbs, thus exiting the state diagram.

For EEs:

o10-14: If the CI supports RD Service, any incoming messages which target an EE in the Init state **must** be silently dropped.

- It is an error for a Work Request to specify an EE Context in the Init state.

o10-15: If the CI supports RD Service, and a Work Request is submitted by the Consumer to the Send Queue of an RD QP specifying an EE Context in the Init state, the Work Request **shall** be completed in error.

For QPs:

- Work Requests may be submitted to the Receive Queue but incoming messages are not processed.

C10-24: The CI **shall** allow Work Requests to be submitted to a Receive Queue while its corresponding QP is in the Init State.

- It is an error to submit Work Requests to the Send Queue.

C10-25: If a Work Request is submitted to a Send Queue while its corresponding QP is in the Init State, an immediate error **shall** be returned.

- Work Queue processing on both queues is disabled.

C10-26: Incoming messages which target a QP in the Init state **must** be silently dropped.

10.3.1.3 READY TO RECEIVE (RTR)

The characteristics for the Ready to Receive state are:

C10-27: The CI **shall** support posting Work Requests to Receive Queue of a QP in the RTR state.

C10-28: Incoming messages targeted at a QP in the RTR state **shall** be processed normally.

o10-16: If the CI supports RD Service, and an incoming message is addressed to an EE Context in the RTR state, the message **shall** be processed normally.

- Transition into this state is possible only from the Init state, using the Modify Queue Pair or Modify EE Context Attributes Verbs.
- Transition out of the RTR state can be effected by calling the Destroy QP or Destroy EE Context Verbs, thus exiting the state diagram.

For EEs:

- It is an error for a Work Request to specify an EE Context in the RTR state.

o10-17: If the CI supports RD Service, and a Work Request is submitted by the Consumer to the Send Queue of an RD QP specifying an EE Context in the RTR state, the Work Request **shall** be completed in error.

For QPs:

- Work Queue processing on the Send Queue is disabled. It is an error to post Work Requests to the Send Queue.

C10-29: If a Work Request is submitted to a Send Queue while its corresponding QP is in the RTR State, an immediate error **shall** be returned.

10.3.1.4 READY TO SEND (RTS)

Before transitioning to this state, the QP/EE communication establishment protocol must be completed.

The characteristics for the Ready to Send state are:

- The channel between the requester's QP/EE and responder's QP/EE has been established for connected Service Types and RD channels.
- Transition into this state is possible only from the RTR and SQD states.
- The Modify Queue Pair or Modify EE Context Attributes Verbs are the only way for the Verbs Consumer to cause a transition out of the RTS state, without destroying the EE/QP.
- Transition out of the RTS state can be effected by calling the Destroy Queue Pair or Destroy EE Context Verbs, thus exiting the state diagram.

C10-30: The CI **shall** support posting Work Requests to a QP in the RTS state.

C10-31: Work Requests on a QP in the RTS state **shall** be processed normally.

C10-32: Incoming messages targeted at a QP in the RTS state **shall** be processed normally.

o10-18: If the CI supports RD Service, and an incoming or outgoing message utilizes an EE Context in the RTS state, the message **shall** be processed normally.

10.3.1.5 SEND QUEUE DRAIN (SQD)

The characteristics for the Send Queue Drain state are:

C10-33: The CI **shall** support posting Work Requests to a QP in the SQD state.

C10-34: Incoming messages targeted at a QP in the SQD state **shall** be processed normally.

o10-19: If the CI supports RD Service, and an incoming message utilizes an EE Context in the SQD state, the message **shall** be processed normally.

- Transition into this state is possible only from the RTS state, using the Modify Queue Pair or Modify EE Context Attributes Verbs.

C10-35: When transitioning into the SQD state, the QP/EE's send logic **must** cease processing any additional messages. It **must** also complete any outstanding messages on a message boundary, and process any incoming acknowledgements. The CI **must not** begin processing additional messages which had not begun execution when the state transition occurred.

C10-36: When all expected acknowledgements have been received, and processing of send queue work requests has ceased, and if event notification has been requested, an Affiliated Asynchronous Event **shall** be generated.

- The consumer can use the asynchronous event to determine when a state transition is possible.
- It is possible to enter the RTS state or error states from the SQD state via Modify Queue Pair or Modify EE Context Attributes Verbs.
- Attributes may be modified during the transition from SQD to RTS, but both sides must have received the affiliated asynchronous event in order to safely change attributes.

o10-19.a1: If the CI supports the ability to change the physical port associated with an RC QP when transitioning from SQD to RTS, the CI shall associate the physical port, if a different physical port is specified, with the QP before the transition from SQD to RTS has completed. The physical

port is an optional attribute in the Modify QP verb during the transition from SQD to RTS.

o10-19.a2: If the CI supports RD service and supports the ability to change the physical port associated with the EE Context when transitioning from SQD to RTS, the CI shall associate the physical port, if a different physical port is specified, with the EE Context before the transition from SQD to RTS has completed. The physical port is an optional attribute in the Modify EE Context verb during the transition from SQD to RTS.

- It is also possible to transition out of the SQD state by calling the Destroy Queue Pair or Destroy EE Context Verbs, thus exiting the state diagram.

For EEs:

- Work Queue processing on the Send side of the EE Context is disabled.

o10-20: If the CI supports RD Service, Work Requests submitted to the Send Queue of an RD QP, which specify an EE Context in the SQD state, **must** not be processed but **shall** remain enqueued.

- QPs associated with an EE do not transition to the SQD state automatically, nor is it inherently necessary they do so.

For QPs:

- Work Queue processing on the Send Queue is disabled.

C10-37: Work Requests submitted to the Send Queue of a QP in the SQD state **must not** be processed but **shall** remain enqueued.

SQD to SQD state transition can be performed when the SQ draining is completed, i.e. the CI has completed processing any outstanding message on a message boundary and processed any incoming acknowledgements, or while the SQ draining is in progress.

However, changing some of the optional attributes on a SQD to SQD transition if the SQ has not been fully drained is not allowed. The CI shall report an immediate error if the verbs consumer performs a SQD to SQD state transition and attempts to change such an optional attribute while the SQ has not been drained yet. Refer to 11.2.3.2 and 11.2.6.2 for the list of attributes that can not be changed while the SQ is still draining. To determine if the SQ has been drained, Software can use the Affiliated Asynchronous Event or to use the Query QP or Query EE verb.

It is not allowed to perform SQD to RTS state transition if the SQ has not been drained yet. The CI shall report an immediate error if the consumer

attempts to move the QP/EE from SQD to RTS if the SQ has not been drained.

10.3.1.6 SEND QUEUE ERROR (SQER)

The characteristics for the Send Queue Error state are:

- Transition into this state can only happen as the result of a Completion Error, which occurred during the processing of a Work Request on the Send Queue while in the RTS state.
- The transition into the Send Queue Error state applies to all QPs except for RC QPs.

C10-38: Receive Work Requests which were submitted to a Receive Queue prior to that queue's transition into the SQEr state **shall** continue to be processed normally. New Receives **must** be able to be posted to such a Receive Queue. Incoming messages which target a QP in the SQEr state **must** be processed normally.

C10-39: A Work Request which caused the Completion Error leading to the transition into the SQEr state **must** return the correct Completion Error Code for the error through the Completion Queue.

- This WR may have been partially or fully executed, and thus may have affected the state of the receiver, as follows:

  Send operations may have been partially or fully completed; because of this, a completion queue entry may or may not have been generated on the receiver.

  RDMA Read operations may have been partially completed; because of this, the contents of the memory locations pointed to by the data segments of their Work Requests are indeterminate.

  RDMA Write operations may have been partially completed; because of this, the contents of the memory locations pointed to by the remote address of their Work Requests are indeterminate. If the operation specified Immediate Data, a completion queue entry may or may not have been generated on the receiver.

  Atomic operations may, or may not have been attempted; because of this, the contents of the memory locations pointed to by the remote address of the Work Request may have a value consistent with either event. At the local node, the contents of the memory locations pointed to by the data segments of their Work Requests are indeterminate.

C10-40: Work Requests on the Send Queue, subsequent to that which caused the Completion Error leading to the transition into the SQEr state, **must** return the Flush Error completion status through the Completion Queue.

- Depending on the Service Type of the QP, some of the subsequent WRs may have been in progress when the error occurred. This may have affected state on the remote node. The possible effects depend on the WR type as noted above.
- The Modify Queue Pair Verb can be used to transition from the SQEr state to the RTS state.
- The Modify Queue Pair Verb can be used to transition from the SQEr state to the Reset or the Error state.
- A Receive Queue Error or an Asynchronous Error will result in a transition to the Error State.
- Transition out of the SQEr state can be effected by calling the Destroy Queue Pair Verb, thus exiting the state diagram.

10.3.1.7 ERROR

The characteristics for the Error state are:

- Normal processing on the QP/EE is stopped.

C10-41: A Work Request which caused the Completion Error leading to the transition into the Error state **must** return the correct Completion Error Code for the error through the Completion Queue.

- This WR may have been partially or fully executed, and thus may have affected the state of the receiver, as follows:

  Send operations may have been partially or fully completed; because of this, a completion queue entry may or may not have been generated on the receiver.

  RDMA Read operations may have been partially completed; because of this, the contents of the memory locations pointed to by the data segments of their Work Requests are indeterminate.

  RDMA Write operations may have been partially completed; because of this, the contents of the memory locations pointed to by the remote address of their Work Requests are indeterminate. If the operation specified Immediate Data, a completion queue entry may or may not have been generated on the receiver.

  Atomic operations may, or may not have been attempted; because of this, the contents of the memory locations pointed to by the remote address of the Work Request may have a value consistent with either event. At the local node, the contents of the memory locations pointed to by the data segments of their Work Requests are indeterminate.

C10-41.2.1: Incoming messages which target a QP in the Error state must be silently dropped.

C10-42: Work Requests subsequent to that which caused the Completion Error leading to the transition into the Error state, including those submitted after the transition, **must** return the Flush Error completion status through the Completion Queue.

- Depending on the Service Type of the QP, some of the subsequent WRs may have been in progress when the error occurred. This may have affected state on the remote node. The possible effects depend on the WR type as noted above.
- The Modify Queue Pair or Modify EE Context Attributes Verbs, specifying a transition to the Reset state, are the only means to effect a transition from the Error state to the Reset state.
- Transition out of the Error state can also be effected by calling the Destroy Queue Pair or Destroy EE Context Verbs, thus exiting the state diagram.

For EEs:

- If a Work Request is in process when the error occurred, the Work Request is completed with a completion error.

o10-21: If the CI supports RD Service, and an RD Work Request uses an EE context which is in the error state, that WR **must** be completed in error. This **shall** place the Sending QP into the SQEr state.

- Errors that occur on an EE may not have a corresponding effect on the QP state.

For QPs:

- For Affiliated Asynchronous Errors, it may not be possible to continue to process Work Requests. In this case, outstanding Work Requests will not be completed.
- When handling the error notification, it is the responsibility of the Consumer to ensure that all error processing has completed prior to forcing the QP to reset.

10.4 AUTOMATIC PATH MIGRATION

Automatic Path Migration is an optional facility that enables connection recovery in the case of failures. Automatic path migration is available for Reliable and Unreliable Connected QP Service Types and Reliable Datagram EE Contexts.

This section explains Automatic Path Migration from the software transport perspective. A hardware-centric description is contained in the Channel Adapter section, 17.2.8 Automatic Path Migration on page 1031.

The Modify Queue Pair and Modify EE Context Attributes Verbs provide the basic capability to load an alternate path and to transition the path migration states defined in 10.4.1 Path Migration State Diagram.

Automatic path migration is enabled or re-enabled by loading an alternate path on the pair of connected QP or EE Contexts and setting the path migration state to Rearm. The Communication Manager defines protocols and mechanisms, which may be used to enable or re-enable Automatic Path Migration on both the local and the remote, connected QP or EE Context. The Communication Manager support for Automatic Path Migration is described in 12.6 Communication Management Messages on page 655 and in 12.8 Alternate Path Management on page 680.

Once Automatic Path Migration has been enabled on both ends of a connected QP/EE, it is possible for the migration to be initiated by transitioning the QP/EE path migration state from Armed to Migrated either from above or below the Verbs interface. The policy used by the Verbs Consumer or the CI to determine when a path migration should be attempted is outside the scope of the architecture.

10.4.1 PATH MIGRATION STATE DIAGRAM

o10-22: If Automatic Path Migration is supported, the CI **shall** implement the Migrated, Rearm, and Armed path migration states for each Reliable Connected and Unreliable Connected queue pair. Transitions between those path migration states **must** be restricted to those shown in Figure 125.

o10-23: If Automatic Path Migration and Reliable Datagram service are supported, the CI **shall** implement the Migrated, Rearm, and Armed path migration states for each EE Context. Transitions between those path migration states **must** be restricted to those shown in Figure 125.

The path migration states apply to a QP or EE Context, but are only tangentially related to the QP/EE Context states described in 10.3.1 Queue Pair and EE Context States.

o10-24: If Automatic Path Migration is supported, and the Verbs Consumer attempts to change the path migration state from Migrated to Rearm during a transition to a QP/EE state other than RTS, an immediate error **shall** be returned.

o10-25: If Automatic Path Migration is supported, and the Verbs Consumer attempts to change the path migration state from Armed to Migrated during a transition from a QP/EE state other than RTS or SQD, an immediate error **shall** be returned.

The relationship of the path migration states to the communication establishment process is defined in 12.9.7 State and Transition Definitions on page 689.

It is permissible to replace the existing alternate path with a new alternate path while the path migration state is Armed (Automatic Path Migration is enabled). The mechanism to load a new alternate path is the same as the one used to reload an alternate path after a path migration has occurred as described in 12.8: Alternate Path Management.

The path migration states are shown in Figure 125.

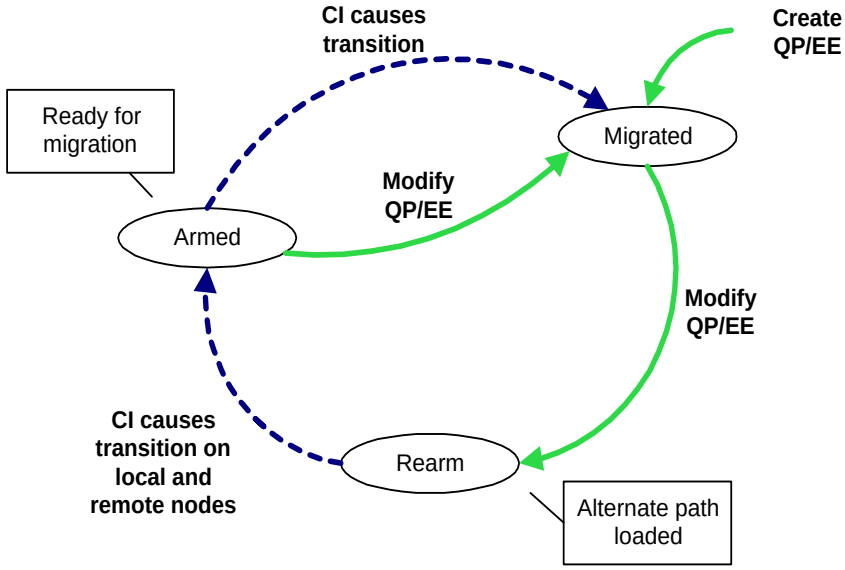


Figure 125 Path Migration State Diagram

10.4.1.1 MIGRATED

o10-26: If Automatic Path Migration is supported, the initial path migration state for a QP/EE **shall** be Migrated.

The Automatic Path Migration capability is suppressed while the state is set to Migrated.

The Verbs Consumer should leave the path migration state for the QP/EE to Migrated under the following circumstances:

- The local CI does not support Automatic Path Migration. If the Verbs Consumer attempts to change the path migration state using the Modify Queue Pair or Modify EE Context Attributes Verbs, an immediate error will be returned.
- The Verbs Consumer does not wish to enable Automatic Path Migration on the QP/EE pair.
- The remote CI does not support or desire Automatic Path Migration. If the Verbs Consumer changes the path migration state to Armed using the Modify Queue Pair or Modify EE Context Attributes Verbs, the path migration state for the QP/EE is changed accordingly and no errors are generated. The local CI shall not transition the QP/EE from Rearm to Armed. Handling this condition is outside of the scope of the architecture.

The Verbs Consumer or the CI may set the path migration state to Migrated when the current path migration state is Armed and the QP/EE state is SQD or RTS. The decision of when to migrate is a matter of policy, which is outside the scope of the architecture.

o10-27: If Automatic Path Migration is supported, a transition from Armed to Migrated **shall** result in a migration to the alternate path on the local QP/EE. The CI **shall** raise the Path Migrated affiliated asynchronous event and **shall** send the next data packet using this QP/EE on the new path with a migration request.

The remote, connected QP/EE validates this request as defined in section 17.2.8 Automatic Path Migration on page 1031.

o10-28: If Automatic Path Migration is supported, upon successfully validating an incoming packet's migration request, the CI **shall** set the path migration state for that QP/EE to Migrated, **shall** migrate to the alternate path, and **shall** also raise the Path Migrated affiliated asynchronous event for that QP/EE.

o10-29: If Automatic Path Migration is supported, upon failing to validate an incoming packet's migration request, the CI **shall not** modify the path migration state for that QP/EE, **shall not** migrate to the alternate path, but **shall** raise the Path Migration Request Failed affiliated asynchronous error for that QP/EE.

The Verbs Consumer should set the path migration state to Migrated only when the current path migration state is Armed and the QP/EE state is SQD or RTS. The Modify Queue Pair or Modify EE Context Attributes Verbs shall generate an immediate error when the Verbs Consumer attempts to set the path migration state to Migrated under any other condition.

o10-30: If Automatic Path Migration is supported, the CI **shall** change the local path migration state to Migrated only when the current state is Armed and the QP/EE state is SQD or RTS.

10.4.1.2 REARM

Only the Verbs Consumer is allowed to initiate the transition from Migrated to Rearm using the Modify Queue Pair or Modify EE Context Attributes Verbs.

o10-31: If Automatic Path Migration is supported, the CI **shall not** change the local path migration state from Migrated to Rearm except at the request of the Verbs Consumer.

The Verbs Consumer should load or reload the alternate path and ensure the remote node has accepted the alternate path prior to transitioning the state from Migrated to Rearm. A transition to the Rearm state indicates to the CI that the Verbs Consumer believes this QP/EE is ready to be transitioned to the Armed state. An invalid or stale alternate path will not generate any errors when the Verbs Consumer transitions the state to Rearm. Handling this condition is outside the scope of the architecture.

o10-32: If Automatic Path Migration is supported, a transition to the Rearm state **shall** cause the CI to attempt to coordinate with the remote, connected QP/EE to move both the local and the remote connected QP/EE into the Armed state in a lock-step manner.

The details regarding how the CIs perform this transition are contained in 17.2.8 Automatic Path Migration on page 1031.

The QP/EEs at both ends of the connection must be in the Rearm state before the CI can transition them to the Armed state.

10.4.1.3 ARMED

The Armed state indicates that the CIs associated with the connected QP/EEs on both the local and the remote node are ready to perform a path migration.

10.5 MULTICAST SERVICES

Multicast is the ability to send a message to a single multicast address and have it delivered to multiple queue pairs which may be on multiple endports. A multicast address is defined by a MGID and a MLID. Multiple MGIDs can be associated with a single MLID. However, a given MGID cannot be associated with more than one MLID on the same subnet. There are two types of multicast specified by IBA: IBA unreliable multicast, and raw packet multicast.

IBA Unreliable Multicast is an optional feature for HCAs. An HCA can be queried to determine the number of multicast groups supported by that HCA. The number of multicast groups is set to zero if the HCA does not support IBA unreliable multicast.

o10-33: If the CI supports IBA Unreliable Multicast, it **must** support at least one multicast group.

Raw packet multicast is an optional feature for HCAs. An HCA can be queried to determine whether it supports raw packet multicast.

10.5.1 MULTICAST GROUPS AND MULTICAST MESSAGE RECEPTION

A multicast group is a collection of endnodes which receive multicast messages sent to a single multicast address. Multicast groups are a fabric management responsibility and are targeted through the use of a multicast address.

10.5.1.1 IBA UNRELIABLE MULTICAST RECEPTION

o10-34: If the CI supports IBA Unreliable Multicast, a UD QP **must** be attached to a multicast group in order to receive IBA Multicast messages.

A QP is attached to or detached from a multicast group through the Verbs. The only function of the Attach QP to Multicast Verb is to assign a receive QP to the multicast group. If the HCA does not have the ability to allow the QP to attach to the multicast group, it shall return an immediate error indicating that there are insufficient resources.

One or more QPs, up to the maximum supported by the HCA, can be attached to each multicast group. In order to receive packets sent to the Multicast group, every QP attached to a particular multicast group should be a member of the same partition as the partition of the incoming packet.

Only Unreliable Datagram QPs can be used for IBA unreliable multicast. Therefore, all Unreliable Datagram semantics also apply to IBA unreliable multicast.

When a valid multicast packet is received successfully, the CI delivers the multicast packet to each UD QP that is associated with the multicast address as defined by the MGID and MLID of the incoming packet.

Note, the incoming multicast packet is received on all the UD QPs associated with the multicast address whether some of those UD QPs share an SRQ or not. That is, if a multicast packet arrives then all UD QPs associated with the packet's multicast address must receive the incoming packet, even if the UD QPs share the same SRQ.

For multiple UD QPs that attach to the same SRQ and have the same multicast address, when a multicast packet arrives that references the multicast UD QPs' multicast address, for each UD QP that is associated with the multicast address on the port the multicast packet was received, the CI must:

- retrieve a WQE from the SRQ; note, if a WQE is not available, then the incoming message will be dropped;
- receive (or copy) the incoming multicast packet into the UD QP's Receive Queue, and
- generate a Work Completion on the CQ associated with the UD QP.

10.5.1.2 RAW PACKET MULTICAST RECEPTION

Raw packet QPs are not attached to multicast groups in order to receive raw packet multicast messages. If an HCA supports only one raw IPv6 QP per port, all raw IPv6 multicast messages received on a port are delivered to that port's raw IPv6 QP; if multiple raw IPv6 QPs are supported, raw IPv6 multicast messages are delivered to a subset of those QPs based on an implementation-defined policy which is outside the scope of IBA. Similarly, if an HCA supports only one raw ethertype QP per port, all raw ethertype multicast messages received on a port are delivered to that port's raw ethertype QP; otherwise, the distribution of those messages is again implementation-defined.

Only raw packet QPs can be used for raw packet multicast. Therefore, all raw semantics also apply to raw packet multicast.

10.5.2 MULTICAST WORK REQUESTS

10.5.2.1 IBA UNRELIABLE MULTICAST WORK REQUESTS

IBA unreliable multicast Work Requests must be submitted through the Post Send Request Verb to a single destination multicast address. This destination multicast address is specified with an Address Handle as part of the Work Request. Any Unreliable Datagram QP can be used to initiate an IBA unreliable multicast Work Request. A QP is not required to be attached to a Multicast Group in order to initiate an IBA Unreliable Multicast Work Request.

Send is the only operation allowed on an Unreliable Datagram Send Work Queue. Atomic and RDMA operations are not allowed. Unreliable Datagram messages should be no larger than the PMTU between the requester and the responder. UD restrictions apply to IBA unreliable multicast.

10.5.2.2 RAW PACKET MULTICAST WORK REQUESTS

Raw packet Multicast Work Requests must be submitted through the Post Send Request Verb to a single destination multicast address. This desti-

nation multicast address is specified as a modifier to the Post Send Request Verb. Any raw packet QP can be used to initiate a raw packet multicast Work Request.

Send is the only operation allowed on a raw packet Send Work Queue. Atomic and RDMA operations are not allowed. Raw packet messages should be no larger than the PMTU between the requester and the responder. Raw restrictions apply to Raw packet multicast.

10.5.3 MULTICAST DESTINATION ESTABLISHMENT

A multicast group is defined by a MGID.

o10-35: If the CI supports IBA Unreliable Multicast, then the CI shall drop all IBA Unreliable Multicast packets if the destination QP number is not 0xFFFFFF.

The special multicast QP number does not have to be the QP number used by the destination to receive a multicast.

C10-43: The method for preparing a multicast address as a destination **shall be** the same as any other address specified in a Work Request on an Unreliable Datagram or Raw Packet Service Type.

Creating & Destroying multicast groups are fabric management issues. Permitting nodes to join and leave a multicast group is a fabric management issue. The MTU of a multicast group is the MTU specified when the multicast group is created and is a parameter in the multicast MAD.

o10-36: If the CI supports IBA Unreliable Multicast, then Multicast loopback, which is sending an IBA unreliable multicast message to a multicast group to which QPs within the sending node are attached, **must** be supported by the CI.

As with multicast reception, loopback for raw packet multicast depends on the number of raw packet queue pairs per port which an HCA supports. If an HCA supports only one raw queue pair of each type per port, then no loopback is performed; multicast messages sent on such a QP are not received on that same QP. If an HCA supports multiple raw QPs of each type per port, then multicast messages sent on one may or may not be received on another; the details of this are an implementation-defined policy which is outside the scope of IBA.

10.6 MEMORY MANAGEMENT

10.6.1 OVERVIEW

The InfiniBand™ Architecture provides sophisticated high performance operations like remote DMA and user mode IO. To achieve this goal, The

InfiniBand™ Architecture has to specify appropriate memory management mechanisms. The overriding goals are performance, robustness and simplicity.

10.6.2 MEMORY REGISTRATION

An HCA, like a typical I/O bus host bridge, accesses Host System memory using what this specification refers to as physical memory addresses[14]. Physical address space for Host System memory is typically organized into pages of fixed or varying sizes, and a given logical data buffer that spans page boundaries usually has a non-contiguous physical address range.

Memory Registration provides mechanisms that allow Consumers to describe a set of virtually contiguous memory locations or a set of physically contiguous memory locations to the Channel Interface in order to allow the HCA to access them as a virtually contiguous buffer using Virtual Addresses.

All Consumers must explicitly register the memory locations containing data buffers before the HCA can access them.

C10-44: This compliance statement has been obsoleted.

C10-44.2.1: If the CI doesn't support the Base Memory Management Extensions defined in this specification and the CI processes a WR or incoming RDMA or Atomic request that attempts to access memory locations that have not been registered, the CI **must** not perform the access, and the CI **must** return an appropriate error.

o10-36.2.1: If the CI supports the Base Memory Management Extensions defined in this specification and the CI:

- processes a WR that attempts to access memory locations that use the Reserved L_Key, but the QP is prohibited from using the Reserved L_Key, then the CI **must** not perform the access, and the CI **must** return an appropriate error;
- processes a Post Send Queue WR that attempts to register memory locations through a Fast Register Physical MR, but the QP is prohibited from using Fast Register Physical MRs, then the CI **must** not perform the access, and the CI **must** return an appropriate error;

14. On some Host Systems, such "physical addresses" are actually mapped by the Host System memory controller to provide features such as memory interleaving or memory sparing, but this specification still refers to them as physical addresses.

- processes an incoming Send, RDMA, or Atomic request that attempts to access memory locations that do not have a valid registration (i.e is in the Invalid or Free State), then the CI **must** not perform the access, and the CI **must** return an appropriate error.

Registration may fail due to unavailability of the necessary Channel Interface resources. No memory is registered in this case.

C10-45: Registration **must** either fully succeed or fail in an atomic fashion.

10.6.2.1 MEMORY REGIONS

A set of memory locations that have been registered are referred to as a Memory Region.

The products of a memory registration operation are:

- MemoryRegionHandle

  The Memory Registration Verbs produce a MemoryRegionHandle that is used to identify a specific Memory Region to the Memory Management Verbs.

- L_Key

  The Memory Registration Verbs produce an L_Key. The L_Key, along with a Virtual Address that is within the bounds of the region is used in a Work Requests's data segment to identify a memory location within a specific Memory Region to the CI.

- R_Key

  The Memory Registration Verbs produce, when requested, an R_Key. The R_Key, along with a Virtual Address that is within the bounds of the region is used in RDMA and Atomic operations to identify a memory location within a specific Memory Region to the CI.

- Virtual Address (VA) and Zero Based Virtual Address (ZBVA)

  The Memory Registration Verbs are supplied (or in some cases produce) either a Virtual Address or a Zero Based Virtual Address. Both correspond to the first memory location in the set of memory locations supplied to the Memory Registration Verbs. However, for a Virtual Address, the first memory location of a Memory Region must have the Virtual Address modulo the buffer size of the first physical buffer equal the first byte offset. For a Zero Based Virtual Address, the first memory location of a MR has a Virtual Address of zero.

When registering a Memory Region, the Consumer specifies the maximum number of memory locations that are to be reserved for future use in Memory Registrations or in Fast Register Physical MRs. For a more de-

tailed description of the attributes associated with the Memory Registration, refer to the various Memory Registration Verbs in Chapter 11.2.8 Memory Management on page 592.

C10-45.2.1: The CI must allow memory registrations and Memory Windows to use Virtual Addresses.

o10-36.2.2: If the CI supports Zero Based Virtual Addresses (Zero Based VAs), the CI **must**:

- allow VA based Memory Windows to be bound to VA based Memory Regions,
- allow Zero Based VA Memory Windows to be bound to VA based Memory Regions, and
- return an error if the Consumer attempts to bind a Memory Window to a Zero Based VA Memory Region.

10.6.2.2 ALLOCATION OF MEMORY REGISTRATION RESOURCES

Allocation of an L_Key allows a Privileged Consumer to reserve, with the CI, memory registration resources for use in future Memory Region Registrations. L_Key allocation is performed through the Allocate L_Key verb.

The products of the Allocate L_Key are:

- MemoryRegionHandle

  A MemoryRegionHandle that is used to identify a specific L_Key to the Memory Management Verbs.

- L_Key

  The Allocate L_Key Verb produces an L_Key. The L_Key is not associated with a Memory Region until it is registered through: Reregister Memory Region, Reregister Physical Memory Region, or Fast Register Physical MR.

- R_Key

  The Allocate L_Key Verb produces, when requested, an R_Key. The R_Key is not associated with a Memory Region until it is registered through: Reregister Memory Region, Reregister Physical Memory Region, or Fast Register Physical MR.

When Allocating Memory Registration resources, the Consumer specifies the maximum number of Physical Buffer List entries (see section 10.6.4.4.1 Physical Buffer lists on page 484) that are to be reserved for future use in Memory Registrations. For a more detailed description of the attributes associated with the Allocate L_Key Verb see section 11.2.8.1 Allocate L_Key on page 593.

10.6.2.3 MEMORY REGION TYPES

There are two types of Memory Regions: Non-Shared Memory Regions and Shared Memory Regions.

- Non-Shared Memory Region
  - A Non-Shared Memory Region has a valid Memory Registration that is associated with a set of physical memory locations which are not shared. A Non-Shared Memory Region is created through the following verbs: Register Memory Region, Register Physical Memory Region, and Fast Register Physical MR. A Non-Shared Memory Region is also created through the Reregister Memory Region or Reregister Physical Memory Region if the translations are changed.
- Shared Memory Region
  - A Shared Memory Region has a valid Memory Registration that is associated with a set of physical memory locations which are shared. Upon successful return from the Register Shared Memory Region verb: the output Memory Region is a Shared Memory Region and the input Memory Region becomes a Shared Memory Region. A Shared Memory Region is converted into a Non-Shared Memory Region through the Reregister Memory Region and Reregister Physical Memory Region if the translations are changed. Changing a Shared MR to a Non-Shared MR does not affect the state of the other MRs that are sharing the PBL.

A Memory Region is used to refer to both a Non-Shared Memory Region and a Shared Memory Region.

Every Memory Region L_Key or R_Key has three possible states:

- Invalid State - No resources have been allocated for the L_Key or R_Key. An L_Key or R_Key in the Invalid State cannot be used to access host memory. An L_Key or R_Key that is in the Invalid State cannot be invalidated through a Send with Invalidate.
- Free State - A Physical Buffer List and PD have been associated with the L_Key or R_Key, but the L_Key or R_Key is not associated with a Memory Region. An L_Key or R_Key in the Free State cannot be used to access host memory. An L_Key or R_Key that is in the Free State can be invalidated through a Send with Invalidate.
- Valid State - The L_Key or R_Key is associated with a Memory Region. An L_Key or R_Key in the Valid State can be used to access host memory. An L_Key or R_Key that is in the Valid State can be invalidated through a Send with Invalidate.

The following table summarizes the states of Memory Regions L_Keys and R_Keys and the operations allowed on each state:

Table 66 Memory Region States Summary

Property / Operation Allowed	Invalid	Free	Valid
Resources Allocated	No	Yes	Yes
Zero length access	Allowed	Allowed	Allowed
Host Memory Access	Prohibited	Prohibited	Allowed
Remote Invalidate	Prohibited	Allowed	Allowed
Fast Register	Prohibited	Allowed	Prohibited
Local Invalidate	Prohibited	Allowed	Allowed

The following table summarizes Memory Regions L_Keys and R_Keys state transitions through the verbs and remote operations (non verbs operations are marked in *italic*):

Table 67 Memory Region State Transitions[a]

State	Entered Through	Exited Through
Invalid	*Initial State* Deregister MR	Allocate L_Key Register MR Register PMR Register Shared MR
Free	Allocate L_Key Local Invalidate *Incoming Send with Invalidate*	Deregister MR Reregister MR Reregister PMR Fast Register
Valid	Register MR Reregister MR Register PMR Reregister PMR Fast Register Register Shared MR	Deregister MR Local Invalidate *Incoming Send with Invalidate*

a. Note: The MR returned as an output from the Reregister MR or Reregister PMR may either be the existing MR or a new MR. If the existing MR is returned, then it is placed in the Valid state. If a new MR is returned, then the existing MR is placed in the Invalid state and the new MR is in the Valid state. The table above refers to the existing MR state transitions when the MR is reused on a Reregister MR or Reregister PMR.

Note, when a Memory Region is successfully invalidated through either a Local Invalidate WR or an incoming Send with Invalidate Operation (Remove Invalidate), both the L_Key and R_Key, if any, associated with the

Memory Region are placed in the Free State. See section 10.6.5 for a description of Invalidated L_Key.

o10-36.2.3: If the CI supports the Base Memory Management Extensions, the CI **must** allow a Non-Shared Memory Region to be used as an input modifier for the following verbs: Reregister Memory Region, Query Memory Region, Reregister Physical Memory Region, Register Shared Memory Region, Bind Memory Window, Post Send Queue Bind Work Request, Fast Register Physical MR, and Deregister MR.

o10-36.2.4: If the CI supports the Base Memory Management Extensions, the CI **must** allow a Shared Memory Region to be used as an input modifier for the following verbs: Reregister Memory Region, Query Memory Region, Reregister Physical Memory Region, Register Shared Memory Region, Bind Memory Window, Post Send Queue Bind WR, and Deregister MR.

o10-36.2.5: If the CI supports the Base Memory Management Extensions, the CI **must** return an error if a Shared Memory Region is used as an input modifier for the Fast Register Physical MR or Local Invalidate.

10.6.3 ACCESS TO REGISTERED MEMORY

C10-46: The CI **shall** support the following access rights: Local Read, Local Write, Remote Read, and Remote Write.

o10-37: If the CI supports Atomic operations, the CI **shall** support the Remote Atomic access right.

10.6.3.1 LOCAL ACCESS TO REGISTERED MEMORY

A Memory Region is always accessible by the local HCA (i.e. a local HCA is an HCA in the same Host system as the Consumer) it was registered with, the type of access allowed depends on the Access Rights assigned to that Memory Region.

C10-47: The CI **shall** automatically include Local Read in every Memory Region's Access Rights.

The Consumer may request that Local Write be assigned to a Memory Region's Access Rights. If desired, policies related to preventing the assignment of Local Write to a Memory Region can be implemented by the Consumer.

10.6.3.2 REMOTE ACCESS TO REGISTERED MEMORY

The Consumer may, in addition to the Local access rights, assign Remote access rights to a Memory Region. Remote access rights are Remote Read, Remote Write and Remote Atomic. Remote access rights are individually selectable and when selected, allow one or more specific opera-

tion types to access the Memory Region. The Consumer is not allowed to assign Remote Write or Remote Atomic to a Memory Region that has not been assigned Local Write.

C10-48: If a Memory Registration specifies Remote Write or Remote Atomic without specifying Local Write, the CI **must** return an Immediate Error.

10.6.3.3 LOCAL ACCESS KEYS

When resources are allocated to hold a set of memory locations an object called an L_Key is created by the CI and returned to the Consumer. The L_Key is associated with the reserved resources that will be used to hold a set of memory locations. It is considered to be a free L_Key, because it is not associated with any Memory Region. The L_Key returned from the Allocate L_Key, can be used as an input modifier for a Fast Register PMR to associate the L_Key with a Memory Region.

An L_Key is also created through the following Memory Registration verbs: Register Memory Region, Reregister Memory Region, Register Physical Memory Region, Reregister Physical Memory Region, and Register Shared Memory Region.

When a set of memory locations are registered, the L_Key used in the registration is associated with the set of registered memory locations. An L_Key that is associated with a specific set of memory locations is considered to be in the valid state. The Fast Register Physical MR does not create an L_Key, it simply associates an existing free L_Key to a set of memory locations.

The L_Key returned from a Memory Registration can be used as an input modifier for a Reregister Memory Region, Reregister Physical Memory Region, Post Send Local Invalidate WR, Bind Memory Window, Post Send Queue Bind Work Request, Shared Memory Region, and Fast Register Physical MR.

o10-37.2.1: If the CI supports the Base Memory Management Extensions defined in this specification, the L_Key format **must** consist of:

- 24 bit index in the most significant bits of the L_Key, and
- 8 bit key in the least significant bits of the L_Key.

o10-37.2.2: If the CI supports the Base Memory Management Extensions defined in this specification, then when an L_Key is created:

- through a successful Allocate L_Key verb invocation, the CI **must** let the Consumer own the key portion of the returned L_Key;

- through a successful Register Memory Region, Reregister Memory Region, or Register Shared Memory Region verb invocation, the CI **must not** let the Consumer own the key portion of the returned L_Key;
- through a successful Register Physical Memory Region or Reregister Physical Memory Region verb invocation, the CI **must** let the Consumer choose between having the key portion of the L_Key owned by the Consumer or owned by the CI.

Note, the index portion of the L_Key is always owned by the CI.

Memory Regions are described to the CI for local access by either:

- a combination of a Virtual Address within that Memory Region and the L_Key that was returned to the Consumer when the region was registered; or
- If the CI supports the ZBVA Extension defined in this specification, a combination of an offset within that Memory Region and the L_Key that was returned to the Consumer when the region was registered.

10.6.3.4 REMOTE ACCESS KEYS

When resources are allocated to hold a set of memory locations and the Consumer selected Remote Access Enablement, an object called an R_Key is created by the CI and returned to the Consumer. The R_Key is associated with the reserved resources that will be used to hold a set of memory locations. It is considered to be a free R_Key, because it is not associated with any Memory Region. The R_Key returned from the Allocate L_Key, can be used as an input modifier for a Fast Register PMR to associate the R_Key with a Memory Region.

A Memory Region R_Key is also created through the following Memory Registration verbs: Register Memory Region, Reregister Memory Region, Register Physical Memory Region, Reregister Physical Memory Region, and Register Shared Memory Region.

When a set of memory locations are registered with Remote Access Rights, the R_Key used in the registration is associated with the set of registered memory locations. An R_Key that is associated with a specific set of memory locations is considered to be in the valid state. The Fast Register Physical MR does not create an R_Key, it simply associates an existing free R_Key to a set of memory locations.

The R_Key returned from a local Memory Registration can be used as an input modifier for a Reregister Physical Memory Region, Post Send Local Invalidate WR, and Fast Register Physical MR. The local Consumer can use an R_Key that was previously exposed by a remote Consumer as an input modifier of a Post Send with Local Invalidate WR.

o10-37.2.3: If the CI supports the Base Memory Management Extensions defined in this specification, the R_Key format **must** consist of:

- 24 bit index in the most significant bits of the R_Key, and
- 8 bit key in the least significant bits of the R_Key.

o10-37.2.4: If the CI supports the Base Memory Management Extensions defined in this specification, then when an R_Key, that is used in Memory Registration verbs, is created:

- through a successful Allocate L_Key verb invocation, the CI **must** let the Consumer own the key portion of the returned R_Key;
- through a successful Register Memory Region, Reregister Memory Region, or Register Shared Memory Region verb invocation, the CI **must not** let the Consumer own the key portion of the returned R_Key;
- through a successful Register Physical Memory Region or Reregister Physical Memory Region verb invocation, the CI **must** let the Consumer choose between having the key portion of the R_Key owned by the Consumer or owned by the CI.

Note, the index portion of the R_Key is always owned by the CI.

Note: For an L_Key that has an associated R_Key, if the R_Key is Consumer owned, then the key portion of the R_Key must equal the key portion of the L_Key. Additionally, either both the R_Key and L_Key are owned by the Consumer; or both are owned by the CI. Ownership cannot be intermixed.

Memory Regions are described to the CI for remote access by either:

- a combination of a Virtual Address within that Memory Region and the R_Key that was returned to the Consumer when the region was registered; or
- if the CI supports the ZBVA Extension defined in this specification, a combination of an offset within that Memory Region and the R_Key that was returned to the Consumer when the region was registered.

10.6.3.5 PROTECTION DOMAINS

A Protection Domain (PD) associates Memory Regions and Queue Pairs. Protection Domains are specific to each HCA. Each Memory Region must be associated with a single Protection Domain. Multiple Memory Regions may be associated with the same PD. Each Queue Pair in an HCA must be associated with a single Protection Domain. Multiple Queue Pairs may be associated with the same PD. Access to Memory Regions described in Work Requests and Remote Operation requests are allowed only when the Protection Domain of the Memory Region and of the Queue Pair that

is processing the request are identical. The setting of protection domains is expected to be controlled by a Privileged Consumer.

10.6.3.6 SCOPE OF ACCESS

Memory is registered for use on a specific HCA. L_Keys and R_Keys are specific to an HCA and do not grant access to the Memory Region by other local HCAs. The CI is not required to enforce that L_Keys or R_Keys associated with one HCA will always result in an error if used with a different HCA.

10.6.3.7 FAST REGISTRATION

When a QP is created, a Privileged Consumer can request that Fast Registration be enabled on the QP. If Fast Registration is enabled, the CI must allow the Consumer to perform Fast Registrations (i.e. Fast Register Physical MRs). If the QP does not have Fast Registration enabled and the Consumer attempts to perform a Fast Registration, the CI must return a Memory Management Operation Error.

A Fast Registration associates a free L_Key with a set of memory locations (see section 10.6.4.4.1 Physical Buffer lists on page 484 for a description of how Physical Buffers Lists are used to reference host memory locations). If the free L_Key has an accompanying free R_Key, and the Consumer requests Remote Access Rights, a Fast Registration associates the R_Key with a set of memory locations.

o10-37.2.5: For an HCA that supports the Base Memory Management Extensions, the CI **must** return an error if:

- the set of memory locations passed in through the Fast Registration are not less than or equal to the resources reserved for memory locations by the verb that created the L_Key used in the Fast Registration;
- the PD associated with the L_Key passed in through the Fast Registration is not the same as the PD associated with the QP performing the Fast Registration;
- QP performing the Fast Registration does not have Fast Registration enabled;
- the L_Key passed in through the Fast Registration is in the Invalid or Valid State; or
- remote access is requested by the Consumer, but the R_Key is not enabled.

o10-37.2.6: If the HCA supports the Base Memory Management Extensions, the Fast Registration **must** take place before any subsequent Work Request on the same Send Queue is started.

Note, the Consumer may rely on having the Fast Registration complete before the HCA begins processing any subsequent WQEs. After successful posting, if the Fast Registration fails, then the QP goes to the Error state and all subsequent WQEs are flushed.

When the Fast Registration completes successfully:

- the L_Key is considered to be in the Valid State, because the Memory Registration was successfully applied to the Free State L_Key; and
- if the Fast Registration:
  - requested an R_Key and the Free State L_Key has an accompanying Free State R_Key, the R_Key is considered to be in the Valid State, because the Memory Registration was successfully applied to the Free State R_Key.
  - did not request an R_Key, but an R_Key is associated with the L_Key (e.g. the Allocate L_Key verb returned an R_Key), then a remote node will not be allowed to access the Memory Region through the R_Key.

When the Fast Registration does not complete successfully, the state of the L_Key, and it's associated R_Key, is unaffected.

10.6.3.8 MULTIPLE REGISTRATION OF MEMORY REGIONS

The same set of memory locations may be registered multiple times, resulting in multiple MemoryRegionHandles, L_Keys and R_Keys. Each Registration is considered a separate and distinct Memory Region and may be independently associated with a Protection Domain.

C10-49: The CI **shall** support independent registration of partially or completely overlapping sets of memory locations.

For cases where it's desired to have multiple registrations of a specific set of memory locations, provision for optimizing the use of Channel Interface resources is provided. See Section 11.2.8.8 Register Shared Memory Region on page 605.

10.6.4 ADDRESSING MEMORY

10.6.4.1 VIRTUAL ADDRESSES ("POINTERS")

Some processor architectures support global virtual address spaces of 80 bits or more. However, the virtual addresses ("pointers") most applications can readily manipulate and supply as parameters are typically either 32 bits or 64 bits, and actually serve as offsets into the handful of processor memory "segments" associated with the process. Thus, the virtual address parameters passed in by Consumers at the Verbs layer must each be interpreted in the proper context of their associated process. The

L_Key or R_Key that accompanies each virtual address parameter helps the CI identify the appropriate context.

The virtual addresses ("pointers") that Consumers manipulate and pass as parameters are referred to simply as *Virtual Addresses* in this specification. The size of the Virtual Addresses used to specify a memory region to be registered and for local memory locations in Work Requests is implementation dependent. The size of Virtual Addresses used to specify remote memory locations in Work Requests is 64 bits.

10.6.4.2 VIRTUAL TO PHYSICAL TRANSLATIONS

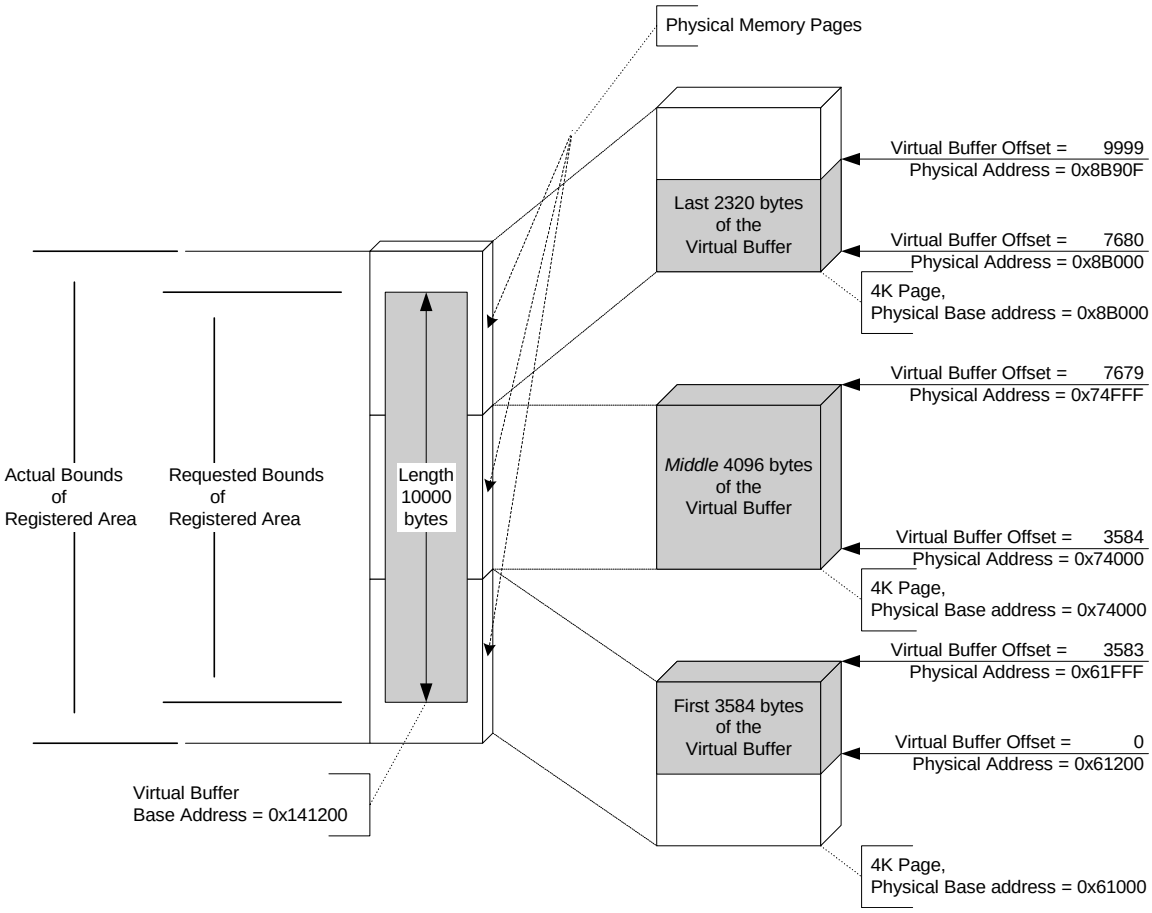


Figure 126 Registered Virtual Buffer to Physical Page Relationship

10.6.4.3 REGISTRATION OF VIRTUALLY ADDRESSED REGIONS

A virtually contiguous set of memory locations are specified by a Virtual Address that points to the first byte of the set and the length of the set in bytes. Figure 126 illustrates an example of a virtually contiguous set of

memory locations backed by three physical pages. The size of the pages that back the region depend on the Host System hardware and Host Operating system.

The pages in the illustration are 4096 bytes each. The actual page size depends on the host hardware and host operating system.

In the example above, access to the memory locations at Virtual Addresses 0x141000 through 0x1411FF may be allowed even though they precede the first address of the region requested to be registered.

10.6.4.3.1 REGISTRATION OF ZERO BASED VIRTUAL ADDRESS (ZBVA) MEMORY REGIONS

The first byte of a ZBVA Memory Region is assigned a virtual address of zero. A ZBVA set of memory locations is specified by an offset from zero, which represents the start of the ZBVA, and the length of the set in bytes. The virtual addresses used in ZBVA Memory Regions do not have any relevant processor context.

10.6.4.3.2 RESERVED L_KEY

If the CI supports the Base Memory Management Extensions, the Consumer can use the Reserved L_Key to identify Post Send and Post Receive Work Request Data Segments. A Data Segment referenced through the L_Key is referenced by a physical address that points to the start of the Data Segment and the length of the Data Segment.

Use of the Reserved L_Key is intended for local use by privileged mode Consumer. When a QP is created, the Consumer must either enable or disable memory access through Fast Register Physical MRs and the Reserved L_Key.

o10-37.2.7: For a CI that supports the Base Memory Management Extensions, if the Consumer enables Fast Register Physical MR and Reserved L_Key access through the Create QP, then on that QP the CI **must** allow the Consumer to:

- use the Reserved L_Key in WRs, and
- use Fast Register Physical MRs.

Note, the Reserved L_Key is a local key and remote access cannot be performed through the Reserved L_Key.

o10-37.2.8: For a CI that supports the Base Memory Management Extensions, the CI **must** return an error if the Consumer attempts to: bind a Memory Window to the Reserved L_Key, Invalidate the Reserved L_Key, destroy the Reserved L_Key, or any type of Registration using the Reserved L_Key.

10.6.4.3.3 BYTE ALIGNMENT AND LENGTH OF MEMORY REGIONS

C10-50: The CI **shall** support arbitrary byte alignment for the virtually contiguous buffer being registered.

C10-51: The CI **shall** support arbitrary length for the virtually contiguous buffer being registered, up to the limit specified by the HCA attribute.

The address translation and access rights of the region applies to each complete physical buffer within that Memory Region.

- If the HCA does not support the Base Memory Management Extensions, the CI is not required to enforce access checks and rights for local or remote accesses with byte-level granularity.
- If the HCA supports the Base Memory Management Extensions, the CI is required to enforce access checks and rights for local and remote accesses with byte-level granularity.

10.6.4.3.4 REGISTERED MEMORY RESIDENCY

C10-52: This compliance statement has been obsoleted.

C10-52.2.1: Using the Verbs defined in this specification, when a Memory Region is registered through a Register Memory Region and Reregister Memory Region, the CI **must** pin down in physical memory every physical buffer within the Memory Region.

Prior to invoking a Register Physical Memory Region or Reregister Physical Memory Region Verb, the Consumer should pin down in physical memory every physical buffer within the Memory Region.

If the HCA supports the Base Memory Management Extensions, when a Memory Region is registered through a Fast Register PMR, the Consumer should pin down in physical memory every physical buffer within the Memory Region.

Pinning down in physical memory every physical buffer within a Memory Region guarantees to the HCA that the Memory Region is physically resident (not swapped out) and that the virtual to physical translation remains fixed while the region is registered.

The Register Memory Region and Reregister Memory Region Verbs are responsible for requesting that the OS pin the associated physical buffers and for requesting from the OS any required per physical buffer Virtual to Physical translation information. The Channel Interface is not required to track physical buffers common to Multiple registrations. The Channel Interface must be able to assume that the OS service that accepts requests for pinning and unpinning physical buffers will maintain the appropriate reference counts on those physical buffers such that pinned physical

buffers are not actually unpinned until the number of unpin requests equal the number of pin requests for any specific physical buffer.

The Register Memory Region and Reregister Memory Region Verbs are expected to request that the OS pin the physical buffers associated with the region every time a region is registered regardless of any association with previously registered regions. The Channel Interface is not prohibited from implementing optimizations that reduce the number of OS service requests it makes for pinning and unpinning memory.

10.6.4.4 REGISTRATION OF PHYSICALLY ADDRESSED REGIONS

As an alternative to specifying a Region by a contiguous range in the Consumer's virtual address space mapped by the processor, Privileged Consumers can specify a Region by a list of physically addressed buffers, which correspond to physical buffers mappable by the HCA. The Consumer also supplies a first byte offset that specifies where the Region begins within the first physical buffer.

There are two types of physically addressed regions: VA based “I/O Virtual Address” and Zero Based “I/O Virtual Address”.

- For VA based IOVA, the Consumer supplies a requested “I/O Virtual Address” to be associated with the first byte of the Region, which is allowed to begin anywhere within the first physical buffer. The Channel Interface returns the I/O Virtual Address that is actually assigned for the Region. A Channel Interface that does not support the Base Memory Management Extensions is not required to assign the I/O Virtual Address requested by the Consumer, but is encouraged to do so wherever possible. A Channel Interface that supports the Base Memory Management Extensions must check that the IOVA modulo the buffer size of the first physical buffer matches the first byte offset and if they match, the CI must assign the IOVA passed in by the Consumer.
- For Zero based IOVA on a Channel Interface that supports the Base Memory Management Extensions, the first byte of the Region, which is allowed to begin anywhere within the first physical buffer, is assigned the address of Zero.

The Consumer also supplies the length of the Region in bytes. The last byte of the Region, as specified by the Region length, must fall within the last physical buffer, but is allowed to fall anywhere within the last physical buffer.

The Virtual Address in this context is called an “I/O Virtual Address” since, depending on the Memory Management Extensions supported by the CI, it may not be mapped in the processor’s virtual address space, and might be used solely for local or remote accesses performed by the HCA.

The Maximum size of an I/O Virtual Address is 64 bits.

10.6.4.4.1 PHYSICAL BUFFER LISTS

C10-53: This compliance statement has been obsoleted.

Two physical buffer lists are supported: page lists and block lists.

Page lists used for registration consist of one or more physically contiguous set of memory locations that must start and end on an CI supported page boundary.

C10-53.2.1: If a page list in a physical memory registration contains an element that does not start and end on a CI-supported page boundary, the CI shall return an error.

o10-37.2.9: Block lists used for registration consist of one or more physically contiguous set of memory locations that may have an arbitrary byte alignment, but must all be of the same size.

The Allocate L_Key verb reserves physical buffer list entries for use in future Memory Region Registrations.

o10-37.2.10: If the CI supports the Base Memory Management Extensions, the CI must return the size reserved for physical buffer list from the following verbs: Allocate L_Key, Query Memory Region, Register Physical Memory Region, and Reregister Physical Memory Region.

All of the physical buffers in a physical buffer list must remain accessible by the CI until after the region has been invalidated or deregistered.

For the case where the physical buffers in the physical buffer list are actually the pinned physical buffers of a virtually addressed buffer, the Consumer is expected to keep those physical buffers pinned while the region is registered.

It is the responsibility of the Consumer to determine if and when, after deregistration the physical buffers should be unpinned. It is the responsibility of the Consumer to ensure proper operation in cases where the physical buffers in the physically addressed region are also in use in a virtually addressed region that has been registered.

10.6.4.5 MEMORY REGION ERROR CHECKING

It is an error for a Consumer to use Virtual Addresses that are outside of the registered locations in a Memory Region.

10.6.4.5.1 ERROR CHECKING OF LOCAL ACCESSES TO MEMORY REGIONS

C10-54: This compliance statement has been obsoleted.

o10-37.2.11: For the Reserved L_Key, the CI **must** ensure that accesses through the Reserved L_Key are enabled on the QP.

C10-54.2.1: If the CI doesn't support SRQ, the CI is **required** to ensure that the memory locations being referenced using a Virtual Address and L_Key are within a physical buffer of a Memory Region with the same PD as the QP that is processing the WR.

o10-37.2.12: If the CI supports SRQ,

- for a QP that is not associated with an SRQ or for a Send Queue of a QP that is associated with an SRQ, the CI is **required** to ensure that the memory locations being referenced using a Virtual Address and L_Key are within a physical buffer of a Memory Region with the same PD as the QP that is processing the WR, and
- for a Receive Queue of a QP that is associated with an SRQ, the CI is **required** to ensure that the memory locations being referenced using a Virtual Address and L_Key are within a physical buffer of a Memory Region with the same PD as the SRQ.

The CI is allowed to support finer-level granularity of local access control.

Generally, the CI is expected to perform validity checking of the Virtual Address and appropriate memory key, L_Key or R_Key, for each Data Segment specified in the Work Request. The exceptions to this rule are as follows:

C10-54.1.1: The CI **shall not** perform any protection checks for zero-length Data Segments.

C10-54.1.2: For UD messages that do not contain a GRH, UC messages, RC messages, and RD messages, the CI **shall not** perform any local protection checking for any Data Segments specified in the Work Request on the receipt of a zero-length message. For UD messages that contain a GRH, the CI **shall** perform the Data Segment checks and, if the check pass, surface the GRH. If the checks don't pass, a completion error shall be surfaced.

C10-55: The CI is **required** to ensure that the Local Access Rights of that Memory Region allow the type of access requested.

It is strongly encouraged that the Channel Interface check and ensure that the Virtual Address is within the Memory Region to which the L_Key is associated and report any bounds violation at access time. It is not mandatory that the Channel Interface enforce such checking.

10.6.4.5.2 ERROR CHECKING OF REMOTE ACCESSES TO MEMORY REGIONS

C10-56: The CI is **required** to ensure that the memory locations being referenced using a Virtual Address and R_Key are within a Memory Region with the same PD as the QP that is processing the Remote Operation. The CI **shall** enforce this with a granularity not to exceed 4096 bytes.

C10-57: The CI is **required** to ensure that the Remote Access Rights of that Memory Region allow the type of access requested.

It is strongly encouraged that the Channel Interface check and ensure that the Virtual Address is within the Memory Region to which the R_Key is associated and report any bounds violation at access time. It is not mandatory that the Channel Interface enforce such checking.

10.6.5 INVALIDATION OF MEMORY REGIONS

When the Consumer no longer needs to access a Physical Memory Region, but wishes to retain the L_Key, and the R_Key (if an R_Key accompanied the L_Key), for use in future Memory Registrations, the Consumer may invalidate the Memory Region, through:

- a local Post Send Local Invalidate WR, or
- a remote Send with Invalidate Operation.

After a local or remote invalidation operation completes successfully, the L_Key is placed in the Free State. In this state the L_Key can no longer be used to access local memory, but it still has physical buffer list resources allocated.

Similarly, after a local or remote invalidation operation completes successfully, the R_Key is placed in the Free State. In this state the R_Key can no longer be used to access local memory, but it still has physical buffer list resources allocated.

For local Invalidates, the Consumer must supply the L_Key, R_Key, and Memory Region Handle for the Memory Region that is to be invalidated. The CI may use any of the three to perform the invalidation operation. The CI is not expected to perform consistency checks between these three input modifiers. For outbound Send with Invalidate, the Consumer must supply the remote R_Key for the remote Memory Region that is to be invalidated.

Memory locations that have been registered multiple times will be represented by multiple Memory Regions. The invalidation of single Memory Region prevents HCA access to those memory locations via the L_Key (and accompanying R_Key if any) associated with that Memory Region. Access to the memory locations via L_Keys and R_Keys associated with other Memory Regions is not affected.

o10-37.2.13: If the Base Memory Management Extensions are supported, the CI **must**:

- support local Local Invalidate WRs and incoming Send with Invalidates;
- return an error if an invalidation operation is attempted on a Memory Region created through the: Register Memory Region; Reregister Memory Region, and Register Shared Memory Region verbs;
- return an error if an invalidation operation is attempted on a Memory Region that was used as an input modifier to a successful Register Shared Memory Region;
- support independent invalidation of partially or completely overlapping Registered Memory Regions.

o10-37.2.14: If the Base Memory Management Extensions are supported, Work Requests or Remote Operation requests that are in process and actively referencing memory locations in a Memory Region that is invalidated **must** fail with a protection violation.

o10-37.2.15: If the Base Memory Management Extensions are supported, Work Requests that attempt to invalidate a Shared Memory Region **must** fail with a Memory Management Operation Error.

o10-37.2.16: If the Base Memory Management Extensions are supported, incoming Send with Invalidate operations that attempt to invalidate a Shared Memory Region **must** fail.

o10-37.2.17: If the Base Memory Management Extensions are supported, local and remote invalidation of a Memory Region **must** fail if the Memory Region still has Memory Windows bound to it.

o10-37.2.18: If the Base Memory Management Extensions are supported, Work Requests or Remote Operation requests that attempt to access memory locations in a Memory Region that has been invalidated **must** fail with a protection violation.

Note, invalidation operation are allowed only on Non-Shared Physical Memory Regions. Therefore the Consumer should negotiate whether the R_Key associated with the MR can be invalidated or not.

The following table summarizes the basic rules for Fast-Registration and Invalidation of Memory Regions:

Table 68 Non-Shared Memory Regions Invalidation and Fast-Registration Rules

Key Ownership	Created Through	Local and Remote Invalidation	Fast Registration
CI	Register MR Reregister MR	Not Allowed	Not Allowed
Consumer	Allocate L_Key Register Physical MR Reregister Physical MR	Allowed	Allowed
CI	Register Physical MR Reregister Physical MR	Allowed	Not Allowed

10.6.5.1 INVALIDATION ORDERING

Two levels of local invalidation ordering are allowed: Relaxed ordered and Strongly ordered.

o10-37.2.19: Relaxed ordered - The invalidate operation may take place at any time after it has been posted to the Send Queue but **must** take place before a completion is generated and before any subsequent WQE has begun execution.

o10-37.2.20: If the CI supports Local Invalidate Fencing: Strongly ordered - The Invalidate operation **must** take place after all preceding WQEs on the same Send Queue have completed and before the next WQE (immediately after the Invalidate) on the same Send Queue begins execution.

o10-37.2.21: For incoming Send with Invalidate - The operation may take place anytime between the reception of the message and the completion of the message. The invalidate operation **must** be performed after all previous non RDMA Read messages have completed processing. The Invalidate operation does not have any ordering with subsequent messages that target the same connection. Note, subsequent messages that target the same connection may execute partially or completely before the invalidation takes place.

Note, when an outbound Send with Invalidate operation completes at the local CA, there is no guarantee that the Remote Invalidate completed successfully at the remote CA. However, when a Send Data Received is returned through a Work Completion and the Send operation was a Send with Invalidate, the (incoming) Invalidate is guaranteed to have been completed.

o10-37.2.22: If the CI supports the Base Memory Management Extensions, then for Non-Shared PMRs the CI **must** support Relaxed Invalidation Ordering and does not have to support Local Invalidate Fencing.

o10-37.2.23: If the CI supports Local Invalidate Fencing, then for Non-Shared PMRs the CI **must** support:

- Strong Invalidation Ordering when the Consumer sets the Local Invalidate Fence on an Local Invalidate WR; and
- Relaxed Invalidation Ordering when the Consumer does not specify the Local Invalidate Fence on an Local Invalidate WR.

o10-37.2.24: For Non-Shared PMRs, if the Responder CI supports the Base Memory Management Extensions and the R_Key check fails on an incoming Send with Invalidation operation, then at the Responder:

- the R_Key **must not** be invalidated;
- the incoming Send with Invalidate **must** generate a Completion Error, and
- the associated QP **must** be placed in the Error State.

10.6.6 DEREGISTRATION OF REGIONS

When access to a Memory Region by a CI is no longer required, the Consumer may reverse the registration process for that region. The process of deregistering a Memory Region will revoke all HCA access rights to that Memory Region.

Memory locations that have been registered multiple times will be represented by multiple Memory Regions. The deregistration of single Memory Region prevents HCA access to those memory locations via the L_Key (and R_Key if any) associated with that Memory Region. Access to the memory locations via L_Keys and R_Keys associated with other Memory Regions is not affected.

C10-58: The CI **shall** support independent deregistration of partially or completely overlapping Registered Memory Regions.

C10-59: Work Requests or Remote Operation requests that are in process and actively referencing memory locations in a Memory Region that is deregistered **must** fail with a protection violation.

C10-60: Work Requests or Remote Operation requests that attempt to access memory locations in a Memory Region that has been deregistered **must** fail with a protection violation.

The Verbs that cause a Memory Region to be deregistered are expected to request that the OS unpin the pages associated with the region if a re-

quest to pin those physical buffers was performed when the region was registered, regardless of any association with previously registered regions. The Channel Interface is not prohibited from implementing optimizations that reduce the number of OS service requests it makes for pinning and unpinning memory.

10.6.7 MEMORY ACCESS CONTROL

The immediate Consumer of every memory registration related Verb is privileged code in the OS. In general, the OS is responsible for determining and enforcing access control policy for memory registrations it does on behalf of User-level Consumers. For instance, it is anticipated but not required that OSs will enforce policies similar to the following:

- A User-level Consumer has control over which of its memory areas can be accessed by HCA data transfer operations.
- A User-level Consumer can enable any local memory area it has access to for access by HCA data transfer operations.
- A User-level Consumer cannot enable HCA read access to memory areas that the Consumer itself doesn't have read access to.
- A User-level Consumer cannot enable HCA write access to memory areas that the Consumer itself doesn't have write access to.

When a Consumer creates QPs or CQs (through the appropriate Verbs), the HCA driver automatically allocates and pins any local memory needed for the associated control structures. Access by the HCA to these control structures is implicitly enabled. Access by the Consumer to these control structures is supported only indirectly through Verbs, and any Region Handles or L_Keys (if they exist) for the control structures are not exposed to the Consumer.

A Consumer controls which QPs can access which Memory Regions and which Memory Windows through the use of Protection Domains (PDs). Prior to creating any QPs, registering any Memory Regions, or allocating any Memory Windows, the Consumer will allocate one or more PDs. Then, when creating QPs, registering Memory Regions, or allocating Memory Windows, the Consumer specifies which PD each is associated with. QPs can only access Memory Regions or Memory Windows that are in the same PD.

10.6.7.1 LOCAL ACCESS CONTROL

With Sends and Receives, the Consumer explicitly specifies the buffers that are accessed through the local Data Segments it passes in the associated Work Requests. Each local Data Segment contains an address, its associated L_Key, and a length parameter. Multiple local Data Segments can be supplied for each send or receive where scatter/gather operation is desired.

Local Data Segments are also used for RDMA Write gather lists, RDMA Read scatter lists, and AtomicOp return values. Again each *local* Data Segment contains an L_Key which governs local access to the corresponding local Memory Region. However, the *remote* Data Segment associated with an RDMA Write, RDMA Read, or AtomicOp will contain an R_Key instead of an L_Key. This is discussed further below.

Two types of *local* access, read and write, are associated with Memory Regions. Send buffers and RDMA Write gather buffers require local read access. Receive buffers, RDMA Read scatter buffers, and AtomicOp return buffers require local write access.

Though memory registration is required to enforce local access only to page-level granularity, the local Data Segments used by Sends, Receives, RDMA Writes, RDMA Reads, and AtomicOps specify byte starting addresses and byte-count lengths. Thus the Consumer still has byte-level granularity of access control for local buffers accessed by these locally initiated operations. The Consumer can determine the actual range of access control enforced using the Query Memory *Region* Verb.

10.6.7.2 REMOTE ACCESS CONTROL

When a Consumer wants to allow remote agents to access its local memory using RDMA Writes, RDMA Reads, or AtomicOps, the Consumer must explicitly enable *remote* access and pass an appropriate *R_Key* to the remote agent for it to use when initiating these operations that target the Consumer's (local) memory.

A Consumer can use either of two mechanisms to enable remote access to its memory. The first mechanism involves enabling remote access when a Memory Region is registered. The second mechanism involves first allocating and then binding a Memory Window to an existing Memory Region. Either mechanism results in an R_Key with associated remote access rights for a specified memory area.

Three types of *remote* access — read, write, and atomic — are supported. RDMA Write requires write access at the remote target, RDMA Read requires read access at the remote target, and AtomicOps require atomic access at the remote target. While perhaps not obvious, it may make sense for a Consumer to allow atomic access but not allow write access, since AtomicOps are not required by the architecture to be atomic with respect to RDMA writes.

10.6.7.2.1 REMOTE ACCESS DIRECTLY WITH MEMORY REGIONS

When registering a Memory Region, a Privileged Consumer can generally specify any combination of remote access rights for the Region, including all or none. However, if a registration request does not specify local write

access to the region, the CI will return an error if remote write or remote atomic access is specified.

If any remote access rights are specified, the Verb will return an R_Key. This R_Key grants the specified remote access rights for the entire Memory Region as bounded by the byte starting address and byte length, but the granularity of the access control actually enforced by the Channel Interface is allowed to be up to 4096 bytes. The Consumer can determine the actual range of access control enforced using the Query Memory *Region* Verb. It is strongly encouraged that the Channel Interface enforce access control with byte-level granularity.

10.6.7.2.2 REMOTE ACCESS THROUGH MEMORY WINDOWS

When a Consumer needs more flexible control over remote access to its memory, the Consumer can use Memory Windows. Memory Windows are intended for situations where the Consumer:

- wants to grant and revoke remote access rights to a registered Region in a dynamic fashion with less of a performance penalty than using deregistration/registration or reregistration.
- wants to grant different remote access rights to different remote agents and/or grant those rights over different ranges within a registered Region.

To use a Memory Window, the Consumer allocates one and then binds it to a specified address range of an existing Memory Region that is enabled for use with Memory Windows. The range can include the entire Memory Region or any virtually contiguous subset of it. A Memory Window can only be bound to a Memory Region that belongs to the same Protection Domain.

C10-61: The CI **shall** enforce remote access control for Memory Windows with byte-level granularity.

When binding a Memory Window, a Consumer can request any combination of remote access rights for the Window. However, if the associated *Region* does not have local write access enabled and the Consumer requests remote write or remote atomic access for the *Window*, the Channel Interface must return an error either at bind time or access time. See 10.6.7.2.7 Error Checking at Window Bind Time and 10.6.7.2.8 Error Checking at Window Access Time.

C10-62: If a Memory Region does not have local write access enabled, the CI **shall** return an error if a Memory Window Bind request specifies remote write or remote atomic access to that Region. The CI **shall** allow all other requested access rights for Memory Windows.

A Consumer is allowed and commonly expected to enable remote access rights when binding a Window that it may not have enabled when it registered the underlying Region — provided it doesn't violate the above rule regarding local write access. For example, a Consumer might register a Region with no remote access rights, and later bind one or more Windows to that Region that obviously would grant remote access rights.

Allocating or *deallocating* a Memory Window requires a kernel transition, and thus incurs the associated software overhead. *Binding* a Memory Window is performed with a Work Request posted to a send queue, and thus incurs far less software overhead with typical implementations.

C10-63: This compliance statement has been obsoleted.

C10-64: This compliance statement has been obsoleted.

C10-65: This compliance statement has been obsoleted.

C10-66: This compliance statement has been obsoleted.

When a Memory Window is Allocated through the Allocate Memory Window Verb, if the CI supports the Base Memory Management Extensions, the Consumer selects either a Type 1 Memory Window (defined in section 10.6.7.2.3) or a Type 2 Memory Window (defined in section 10.6.7.2.4). Memory Window, without any qualifier, is used to refer to both Type 1 and Type 2 Memory Windows.

Type 1 Memory Windows are addressed only through Virtual Addresses. Type 2 Memory Windows are addressed through either Virtual Addresses or Zero Based Virtual Addresses (see section 10.6.2.1 Memory Regions on page 470 for a description of VA and ZBVA).

Every Memory Window R_Key has three possible states:

- Invalid State - No resources have been allocated for the R_Key. An R_Key in the Invalid State cannot be used to access host memory. An R_Key that is in the Invalid State cannot be invalidated through a Send with Invalidate.
- Free State - The R_Key is not associated with a Memory Region and cannot be used to access host memory. An R_Key that is in the Free State can be invalidated through a Send with Invalidate. Note: The Free state may not be implemented on Type 1 Memory Windows (see Table 69).
- Valid State - The R_Key is associated with a Memory Region. An R_Key in the Valid State can be used to access host memory. An R_Key that is in the Valid State can be invalidated through a Send with Invalidate.

The following table summarizes the states of Type 1 Memory Windows R_Keys and the operations allowed on each state:

Table 69 Type 1 Memory Windows States Summary

Property / Operation Allowed	Invalid	Free[a]	Valid
Resources Allocated	No	Yes	Yes
Bound to a Memory Region	No	No	Yes
Zero length access	Allowed	Allowed	Allowed
Host Memory Access	Prohibited	Prohibited	Allowed
Send with Invalidate	Prohibited	Prohibited	Prohibited
Local Invalidate	Prohibited	Prohibited	Prohibited
Post Send Bind MW	Prohibited	Prohibited	Prohibited
Bind Memory Window	Prohibited	Allowed	Allowed

a. The Free state may not be implemented on Type 1 Memory Windows, instead, the Valid state with the Memory Window length of zero can be used. For this type of implementation, an Unbound MW can be considered to be in the Free State.

The following table summarizes the states of Type 2 Memory Windows R_Keys and the operations allowed on each state:

Table 70 Type 2 Memory Windows States Summary

Property / Operation Allowed	Invalid	Free	Valid
Resources Allocated	No	Yes	Yes
Bound to a Memory Region	No	No	Yes
Zero length access	Allowed	Allowed	Allowed
Host Memory Access	Prohibited	Prohibited	Allowed
Send with Invalidate	Prohibited	Allowed	Allowed
Local Invalidate	Prohibited	Allowed	Allowed
Post Send Bind MW	Prohibited	Allowed	Prohibited
Bind Memory Window	Prohibited	Prohibited	Prohibited

The following table summarizes Type 2 Memory Windows R_Keys state transitions through the verbs and remote operations (non verbs operations are marked in *italic*):

Table 71 Type 2 Memory Windows State Transitions

State	Entered Through	Exited Through
Invalid	*Initial State* Deallocate MW	Allocate MW
Free	Allocate MW Local Invalidate *Incoming Send and Invalidate*	Deallocate MW Post Send Bind MW
Valid	Post Send Bind MW	Deallocate MW Local Invalidate *Incoming Send with Invalidate*

10.6.7.2.3 TYPE 1 MEMORY WINDOWS

Type 1 Memory Windows are Bound through the Bind Memory Window Verb. The CI owns the R_Key associated with a Type 1 Memory Window.

Type 1 Memory Windows have the same PD access control semantics as defined in the InfiniBand™ Architecture Specification, Volume 1, Release 1.1.

C10-66.2.1: An HCA **must** support Type 1 Memory Windows.

C10-66.2.2: Each time a given Type 1 Memory Window is bound through the Bind Memory Window Verb, the CI **shall** return an R_Key whose value is different from the immediate previous value. After the bind operation completes, any access attempts using the immediate previous R_Key **must** fail.

When the Type 1 Memory Window is bound through the Bind Memory Window, the Verb returns the new R_Key immediately after posting the Work Request, even though the actual binding operation performed by the HCA hasn't yet occurred.

> Implementation Note: an envisioned implementation for an R_Key is to have it consist of two fields—an index field and a key field. The index field is used by the HCA to identify the associated Type 1 Memory Window resource, and remains constant. The key field is changed each time the R_Key is bound, which guarantees that the immediate previous R_Key is invalidated as required. The use of a sufficient size key field and suitable random number with each binding can provide some amount of protection against the holder of an inval-

idated R_Key being able to access the Type 1 Memory Window without authorization.

The Channel Interface software that prepares the Bind Work Request generates the new key value and places it in the Work Request for the HCA to record in its Type 1 Memory Window resource when processing the request. This way, the new R_Key value is fully determined and can be returned to the Consumer prior to the HCA processing the request.

It is not required that Channel Interfaces use this implementation.

For correct operation, a Consumer must ensure that no remote agent attempts to use a new R_Key before its associated binding has been completed by the HCA. One technique to accomplish this is for the Consumer to submit the Bind operation to the same Send Queue it uses to send the message that conveys the new R_Key to the remote agent.

The Bind operation has a unique ordering rule:

C10-66.2.3: Any Work Request posted to a Send Queue subsequent to an invocation of the Bind Memory Window Verb **shall** not begin execution until the Bind operation completes.

o10-37.2.25: If an HCA supports the Base Memory Management Extensions, any Work Request posted to a Send Queue subsequent to an invocation of the Bind Memory Window Verb **shall not** begin execution until the Bind operation completes. Local Invalidate WRs posted subsequent to a Bind Memory Window Verb are an exception (see section 10.8.3.3 Send Queue Ordering Rules on page 516).

If the HCA detects an error with the Bind operation, it will put the QP into an error state. With the technique described earlier, the Bind operation is guaranteed to complete before the remote agent can possibly receive the new R_Key.

An envisioned common usage model is for a Type 1 Memory Window to be allocated once and then used for multiple bindings. When a previously bound Type 1 Memory Window is bound again, the previous R_Key and its associated bindings are automatically invalidated. Any remote agents needing to use the new Type 1 Memory Window bindings must use the new R_Key.

If the Consumer wants to invalidate a Type 1 Memory Window's bindings without deallocating the Type 1 Memory Window or enabling remote access to new areas, the Consumer can submit a Bind request specifying a length of zero.

C10-66.2.4: After a zero-length Type 1 Memory Window Bind completes, the CI **shall not** allow any remote access to be performed to that Type 1 Memory Window until a subsequent Bind re-enables remote access.

C10-66.2.5: The CI **shall** support multiple Type 1 Windows bound to the same Memory Region, each with independent remote access rights, and their associated areas **shall** be allowed to be overlapping or disjoint.

o10-37.2.26: For an HCA that supports the Base Memory Management Extensions, Local Invalidate WRs **shall not** allowed on Type 1 Memory Windows.

o10-37.2.27: For an HCA that supports the Base Memory Management Extensions, if an incoming Send with Invalidate targets an R_Key associated with a Type 1 Memory Window, the CI **shall**:

- surface a Memory Management Operation completion error;
- not affect the state of the Type 1 Memory Window; and
- place the QP in the Error state.

10.6.7.2.4 TYPE 2 MEMORY WINDOWS

o10-37.2.28: If the CI supports the Base Memory Management Extensions defined in this specification, the CI **must** support Type 2 Memory Windows.

Type 2 Memory Windows are Bound through the Post Send Bind Memory Window Work Request.

Two types of access controls are defined for Type 2 Memory Windows:

o10-37.2.29: If an HCA supports Type 2A Memory Window, then it **must** provide the following semantics for a Type 2A MW:

- Post Send Bind WRs are allowed if the Type 2A MW is in the Free state and the QP performing the Bind WR is associated with the same PD as the PD associated with R_Key referenced in the Bind WR.
- Invalidate operations are allowed if the Type 2A MW is in the Valid state and the QP Number (QPN) of the QP performing the Invalidate operation matches the QPN associated with the Bound Type 2A MW. Invalidate operations are also allowed if the Type 2A MW is in the Free state and the QP performing the Invalidate operation is associated with the same PD as the PD associated with the R_Key referenced in the Bind WR.

- MW access operations (i.e. RDMA Write, RDMA Reads, and Atomics) are only allowed if the Type 2A MW is in the Valid state and the QP Number (QPN) of the QP performing the MW access operation matches the QPN associated with the Bound Type 2A MW.
- A QP can be destroyed or placed in the Reset State only if it has no Type 2A MWs associated with the QP.

o10-37.2.30: If an HCA supports Type 2B Memory Window, then it **must** provide the following semantics for a Type 2B Memory Window:

- Post Send Bind WRs are allowed if the Type 2B MW is in the Free state and the QP performing the Bind WR is associated with the same PD as the PD associated with R_Key referenced in the Bind WR.
- Invalidate operations are allowed if the Type 2B MW is in the Valid state and the QP Number (QPN) and PD of the QP performing the Invalidate operation matches the QPN and PD associated with the Bound Type 2B MW. Invalidate operations are also allowed if the Type 2B MW is in the Free state and the QP performing the Invalidate operation is associated with the same PD as the PD associated with the R_Key referenced in the Bind WR.
- MW access operations (i.e. RDMA Write, RDMA Reads, and Atomics) are only allowed if the Type 2B MW is in the Valid state and the QP Number (QPN) and PD of the QP performing the MW access operation matches the QPN and PD associated with the Bound Type 2B MW.
- A QP can be destroyed or placed in the Reset State, even if it still has Type 2B MWs associated with it.

o10-37.2.31: If an HCA supports the Base Memory Management extensions, the HCA **shall** support either Type 2A or Type 2B MWs, but not both.

In this specification, the term Type 2 MW is used to describe semantics that are common to both Type 2A and Type 2B MWs.

A Type 2 MW that is in the Invalid State, cannot be used in a Bind or Invalidate operation.

o10-37.2.32: If the CI supports the Base Memory Management Extensions defined in this specification, the R_Key format for a Type 2 Memory Window **must** consist of:

- 24 bit index in the most significant bits of the R_Key, which is owned by the CI, and
- 8 bit key in the least significant bits of the R_Key, which is owned by the Consumer.

o10-37.2.33: If the CI supports the Base Memory Management Extensions, each time a given Type 2 Memory Window is bound through the Post Send Bind Memory Window Work Request, the CI **shall** use the key value provided by the Consumer for the bind operation. After the bind operation completes, the R_Key **must** consist of the 24 bit index associated with the Type 2 Memory Window and the 8 bit key supplied by the Consumer in the Post Send Bind Memory Window Work Request.

o10-37.2.34: If the CI supports the Base Memory Management Extensions, the CI **must** return an error (either an immediate error or a completion error) if the Consumer attempts to use a Type 1 MW R_Key on a Post Send Bind Memory Window Work Request (i.e. on a Type 2 MW).

o10-37.2.35: If the CI supports the Base Memory Management Extensions, the CI **must** return an error (either an immediate error or a completion error) if the Consumer attempts to use a Type 2 MW R_Key on a Bind Memory Window verb (i.e. on a Type 1 MW).

When a Type 2 Memory Window is bound through the Post Send Bind Memory Window Work Request, the CI does not return the R_Key. Note: it is the Consumer's responsibility to remember the R_Key index generated by Allocate Memory Window and construct R_Key values for transmission to the remote Consumer by combining the 24 bit index and the Consumer-chosen 8 bit key.

For correct operation, a Consumer must ensure that no remote agent attempts to use a new R_Key before its associated binding has been completed by the HCA. One technique to accomplish this is for the Consumer to submit the Bind operation to the same Send Queue it uses to send the message that conveys the new R_Key to the remote agent.

The Bind MW and Post Send Bind WR have a unique ordering rule:

o10-37.2.36: Any Work Request posted to a Send Queue subsequent to a Post Send Bind WR **shall not** begin execution until the Post Send Bind WR completes. Local Invalidate WRs posted subsequent to a Post Send Bind WR are an exception (see section 10.8.3.3 Send Queue Ordering Rules on page 516).

If the HCA detects an error with the Bind operation, it will put the QP into an error state. With the technique described earlier, the Bind operation is guaranteed to complete before the remote agent can possibly receive the new R_Key.

An envisioned common usage model is for a Type 2 Memory Window to be allocated once and then used for multiple bindings. Before a Type 2 Memory Window can be bound again, it must be invalidated through either a local invalidate or a remote invalidate.

- The Local Invalidate WR is used to perform a local invalidate on a Type 2 Memory Window.
- An incoming Send with Invalidate Operation is used to perform a remote invalidate on a Type 2 Memory Window.

The same levels of invalidation ordering area allowed on Type 2 Memory Window invalidation as on Memory Regions (see section 10.6.5.1 Invalidation Ordering on page 488 for a description of Invalidation ordering).

Note, when an outbound Send with Invalidate operation completes at the local HCA, there is no guarantee that the Remote Invalidate completed successfully at the remote CA. However, when a Send Data Received is returned through a Receive Work Completion and the incoming Send operation was a Send with Invalidate, the (incoming) Invalidate is guaranteed to have been completed.

o10-37.2.37: If the CI supports the Base Memory Management Extensions, then the CI **must** support Relaxed Invalidation Ordering for Type 2 MWs.

o10-37.2.38: If the CI supports Strong Invalidation Ordering, then for Type 2 MWs the CI **must**: support Strong Invalidation Ordering, and provide an Local Invalidate Fence bit for the Consumer to choose between Relaxed or Strong Invalidation Ordering on Local Invalidates.

o10-37.2.39: For a Type 2 MW, if the CI supports the Base Memory Management Extensions and the R_Key check fails on an incoming Send with Invalidation operation:

- the R_Key **must not** be invalidated;
- the incoming Send with Invalidate **must** generate a Completion Error, and
- the QP **must** go to the Error state.

o10-37.2.40: For a Type 2 MW, if the CI supports the Base Memory Management Extensions and the Consumer attempts to Invalidate a Type 2 Memory Window through either the invocation of a Bind Memory Window or a Post Send Bind WR with a length of zero, the CI **must** return an error.

o10-37.2.41: If the CI supports the Base Memory Management Extensions, the CI **shall** support multiple Type 1, Type 2, or both Type 1 and Type 2 Memory Windows bound to the same Memory Region, each with independent remote access rights, and their associated areas **shall** be allowed to be overlapping or disjoint.

Note, invalidation operation are allowed on Type 2 Memory Windows, but not on Type 1 Memory Windows. Therefore the local Consumer should

negotiate with the remote Consumer whether the R_Key associated with the MW can be invalidated or not.

The following tables depict the differences between type 2A and type 2B MWs. Other rules associated with Type 2A and Type 2B MWs also apply.

Table 72 Post Send Bind WR Rules

MW Type	MW State	PD Rule
Type 2A	Free	PD associated with R_Key in the Post Send Bind WR must be the same as the PD associated with the QP on which the WR was posted.
Type 2B	Free	

Table 73 Type 2 Memory Window Invalidation Rules

MW Type	MW State	QP Rule	PD Rule
Type 2A	Valid	Invalidate operation only allowed via QP on which MW was bound	N/A
Type 2A	Free	N/A	Invalidate operation only allowed via QP associated with same PD as MW
Type 2B	Valid	Invalidate operation only allowed via QP on which MW was bound	
Type 2B	Free	N/A	

Table 74 Type 2 Memory Window Access Rules

MW Type	MW State	QP Rule	PD Rule
Type 2A	Valid	MW access operations only allowed via QP on which MW was bound	N/A
Type 2B	Valid	MW access operations only allowed via QP on which MW was bound	MW access operations only allowed if the PD associated with the R_Key is the same as the PD associated with the QP

The following table summarizes the basic rules for Memory Windows Invalidation (Note that fast registration is never allowed on Memory Windows):

Table 75 Memory Windows Invalidation Rules

MW Type	Key Ownership	Local and Remote Invalidation	Bind using already Bound MW
Type I	CI	Not Allowed	Allowed
Type II	Consumer	Allowed	Not Allowed

10.6.7.2.5 REBINDING OR DEALLOCATING ACTIVE WINDOWS

Under normal operation, it is improper for a Consumer to deallocate or change the binding of a Memory Window while it is being accessed by a remote agent. However, this can occur if remote agents misbehave, or it can occur under error recovery circumstances.

C10-67: Any Remote Operation requests that are in process and actively using a Memory Window *when its binding is changed* **must** fail with a protection violation.

C10-68: Once the Bind operation has been reported to the Consumer as having completed, the Channel Interface **must** guarantee that no additional accesses can be performed under the immediate previous binding.

C10-69: Any Remote Operation requests that are in process and actively using a Memory Window *when it is deallocated* **must** fail with a protection violation.

C10-70: Once the Deallocate Memory Window Verb completes, the Channel Interface **must** guarantee that no additional accesses can be performed through that Memory Window while it remains deallocated.

10.6.7.2.6 DEREGISTERING REGIONS WITH BOUND WINDOWS

It is an error for a Consumer to deregister or reregister a Memory Region while it still has any Memory Windows bound to it. Such Windows are said to be “orphaned”. The Channel Interface must handle this error case as follows.

The Channel Interface is allowed to detect this error case and return an error without carrying out the deregister or reregister operation.

C10-71: If the CI allows a Memory Region deregister or reregister operation to create orphaned Windows, the CI **must** guarantee that any remote accesses attempted through the orphaned Windows will undergo the ac-

cess checks and enforcement described in 10.6.7.2.8 Error Checking at Window Access Time.

10.6.7.2.7 ERROR CHECKING AT WINDOW BIND TIME

The following checks must be performed at Memory Window "bind time", which is either when the Channel Interface is executing the Bind Memory Window Verb that prepares and queues the associated Work Request, or when the HCA is processing that Work Request.

C10-72: The Channel Interface **must** check and enforce that the Memory Window and QP belong to the same PD.

C10-73: The Channel Interface **must** check and enforce that Memory Windows are allowed to be bound to the specified Memory Region.

C10-74: The Channel Interface **must** check and enforce write permissions with the specified Memory Region, as described in 10.6.7.2.2 Remote Access Through Memory Windows .

C10-75: The Channel Interface **must** perform address bounds checks and PD checks with regard to the specified Memory Region.

10.6.7.2.8 ERROR CHECKING AT WINDOW ACCESS TIME

When the HCA processes an inbound RDMA or Atomic request that accesses a:

C10-76: This compliance statement has been obsoleted.

C10-76.2.1: Type 1 Memory Window, the CI **must** check and enforce that the Memory Window and QP are associated with the same PD.

o10-37.2.42: For Type 2A Memory Windows, the CI **must** check and enforce that the Type 2A MW is in the Valid state and is associated with the QPN of the QP performing the inbound remote operation.

o10-37.2.43: For Type 2B Memory Windows, the CI **must** check and enforce that the Type 2B MW is in the Valid state and is associated with the QPN and PD of the QP performing the inbound remote operation.

C10-77: The Channel Interface **must** check and enforce the address bounds and access rights associated with the Window.

C10-78: This compliance statement has been obsoleted.

C10-79: This compliance statement has been obsoleted.

C10-79.2.1: The Channel Interface **must** check and enforce the access rights associated with each accessed physical buffer.

C10-79.2.2: For any previously undetected error cases where the Consumer orphaned the Window as described in 10.6.7.2.6 Deregistering Regions with Bound Windows on page 502, the Channel Interface **must** check and enforce that any physical buffers accessed are in some Memory Region that belongs to the same PD as the Window.

The Channel Interface is not required to enforce that such physical buffers are necessarily in the same Region to which the Window was bound. Again, it is strongly encouraged that the Channel Interface check and report these error cases at bind or deallocation time instead of access time.

10.6.7.2.9 ERROR CHECKING AT TYPE 2 MEMORY WINDOW INVALIDATE TIME

o10-37.2.44: When the HCA processes an inbound Send with Invalidate request that has an R_Key which references a Memory Window, the CI **must** check and enforce that:

- The Memory Window is a Type 2 Memory Window;
- if the R_Key references a Type 2 MW that is in the Free State, the PD associated with the Type 2 MW matches the PD that is associated with the QP (note, in this case, the invalidate will be treated as a no-op);
- if the R_Key references a Type 2A MW that is in the Valid State, the QPN associated with the type 2A MW matches the QPN of the QP that received the Send with Invalidate operation;
- if the R_Key references a Type 2B MW that is in the Valid State, the PD and QPN associated with the Type 2B MW matches the QPN and PD of the QP that received the Send with Invalidate operation.

10.7 WORK REQUESTS

Work Requests are used to submit units of work to the channel interface. There are different types of work requests supported and are abstracted throughout the Verbs.

How a work request targets its destination is dependent upon the work request and QP type. The target memory location is contained in the work requests's remote node address information (in the case of RDMA and Atomics) or in the remote receive QP WR's scatter/gather list (in the case of Send/Receive). The target QP depends on the QP type. Connected QPs have the destination QP contained in the local QP context. Datagram QPs have the destination QP contained as part of the work request. Raw QPs don't target a specific QP at the destination.

10.7.1 CREATING WORK REQUESTS

Work Requests are the only mechanism available to Consumers to generate work on work queues. Work requests are used only to pass the operation from the Consumer to the CI.

Work Requests are created by the Consumer above the Channel Interface using mechanisms provided by the OSV.

10.7.2 WORK REQUEST TYPES

The operations which may be posted to the Work Queues by the Consumer:

- Send/Receive
- RDMA Write
- RDMA Read
- Atomic Operations
- Bind Memory Window
- Local Invalidate - If the Base Memory Management Extensions are supported.
- Fast Register Physical MR - If the Base Memory Management Extensions are supported.

The Base Queue Management Extensions also provide the Consumer with a mechanism to post a list of Work Requests to the Send Queue, Receive Queue, or Shared Receive Queue.

10.7.2.1 SEND/RECEIVE

C10-80: The CI **shall** support Send and Receive Operations on all Transport Service Types supported on the CI.

Sends must be posted to the Send Queue.

Receives must be posted to the Receive Queue.

C10-81: The responder's Receive QP **shall** consume a Work Request on reception of an incoming send message.

C10-82: The CI **shall** provide segmentation and reassembly for RC and UC Transport Service Types.

o10-38: If the CI supports RD Service, the CI **shall** provide segmentation and reassembly for RD.

o10-38.2.1: If the CI supports the Base Memory Management Extensions, the CI **must** support Send with Invalidate Operations on RC Service QPs.

10.7.2.2 RDMA

There are two types of RDMA: RDMA Read and RDMA Write.

RDMA Read Operations are supported only on the two reliable Transport Service Types—Reliable Connection and Reliable Datagram. RDMA Write Operations are supported on the two reliable Service Types plus the Unreliable Connection Service Type.

C10-83: The CI **shall** support RDMA Read Operations on the RC Transport Service Type.

C10-84: The CI **shall** support RDMA Write Operations on the RC and UC Transport Service Types.

o10-39: If the CI supports RD Service, the CI **shall** support both RDMA Read and Write Operations on the RD Transport Service Type.

RDMA Read and RDMA Write requests are submitted to the Send Queue.

C10-85: The responder's Receive Queue **shall not** consume a Work Request for an incoming RDMA Read.

C10-86: The responder's Receive Queue **shall** consume a Work Request when Immediate Data is specified in a successfully completed incoming RDMA Write.

C10-87: The responder's Receive Queue **shall not** consume a Work Request when Immediate Data is not specified in an incoming RDMA Write or the incoming RDMA Write was not successfully completed.

The target address of an RDMA request is the remote node's virtual address, a valid R_Key and length. The R_Key must be associated either a Memory Region or a Memory Window containing that virtual address.

Queue Pairs and Memory Regions or Memory Windows have RDMA Read attributes and RDMA Write attributes. These attributes are checked at the target end and are not checked at the source end.

C10-88: The CI **shall not** transfer data from an RDMA operation into the target memory unless the RDMA operation is enabled for the target QP.

10.7.2.3 ATOMIC OPERATIONS

IB Atomic Operations are architected as an optional feature to enable high-performance synchronization for distributed applications running on multiple hosts on the IB fabric.

Two operation types are supported: Compare & Swap and Fetch & Add. The operand size for these operations is 64 bits. It is the responsibility of the Channel Interface at the local endnode to do any transformation to match the endnode endian convention.

o10-40: If the CI supports Atomic operations, the CI **shall** support two types of Atomic operations, Compare & Swap and Fetch & Add.

o10-41: If the CI supports Atomic operations, the CI at the local endnode **shall** perform any byte ordering transformation required to match the endian endnode convention.

o10-42: If the CI supports Atomic operations, the CI **shall** implement Fetch & Add using two's complement arithmetic without saving the carry.

o10-43: If the CI supports Atomic operations, the CI **shall** return an error if the remote address of the Atomic operation is not aligned on a 64-bit boundary.

It is up to the Consumer to interpret whether the numbers are signed or unsigned.

Atomic Operations are supported only on the two reliable Transport Service Types—Reliable Connection and Reliable Datagram.

o10-44: If the CI supports Atomic operations, the CI **shall** support Atomic operations for the RC Transport Service Type.

o10-45: If the CI supports Atomic operations and the RD Transport Service Type, then the CI **shall** support Atomic operations for an RD QP.

o10-46: If the CI supports Atomic operations, the CI **shall not** support Atomic operations on any other Transport Service Types other than RC and RD.

Atomic Operation requests are posted to the Send Queue. The Atomic Operation request is made using the Post Send Request Verb. The results are contained in the data segment. The completion status of the request posted to the Send Queue indicates only if the Atomic Operation was successfully attempted. The Consumer must check the result to determine if a conditional operation took place.

o10-47: If the CI supports Atomic operations, the CI **shall** return the results of the operation in the Data Segment.

If an HCA supports atomics, then all atomic operation requests made to that HCA, referencing the same physical memory, are guaranteed to ap-

pear to be serialized with respect to each other. These operations may be directed at one or more queue pairs.

o10-48: If the CI supports Atomic operations, the CI **shall** provide the appearance that all Atomic operation requests made to the same HCA, referencing the same physical memory are serialized with respect to each other.

Atomic operation requests made to an HCA are not guaranteed to be serialized with respect to RDMA operation requests made to it or other HCAs in the system, or with respect to operations performed by other system components such as processors. Because of this behavior, if atomic operations on a particular area of memory are used to implement locks, all accesses to that memory must be done using atomic operations. In particular, it is not safe to use an RDMA read or Send/Receive to see if a lock is held, and it is not safe to use an RDMA write or Send/Receive to clear a lock.

Optionally, some systems may choose to provide a stronger guarantee: that all atomic operation requests made to all HCAs in the system, as well as all atomic operations performed on memory by other system components such as processors, referencing the same physical memory, are guaranteed to appear to be serialized with respect to each other. Again, these operations may be directed at one or several separate queue pairs. The definition of an “atomic operation” as performed by a system component which is not an HCA is implementation-dependent; for instance, a processor might be required to execute a particular instruction to produce an atomic operation.

o10-49: If the CI supports Atomic operations and the system provides Atomic access across the system, the CI **shall** provide the appearance that all Atomic operation requests that reference the same physical memory are serialized with respect to each other.

10.7.2.4 BIND MEMORY WINDOWS

The Bind Memory Window operation associates a previously allocated Memory Window to a specified address range within an existing Memory Region, along with a specified set of remote access privileges.

Bind Operations are supported only on the Reliable Connection, Unreliable Connection, and Reliable Datagram Service Types.

C10-89: The CI **shall** support Bind operations for RC and UC Transport Service Types.

o10-50: If the CI supports RD Service, the CI **shall** support Bind operations for the RD Transport Service Type.

Bind operations must be posted to the Send Queue. Binds affect only local HCA memory mapping resources and do not cause any packets to be issued over the link. No resources at the destination QP are affected.

10.7.2.5 LOCAL INVALIDATE

The Local Invalidate Operation is allowed on Non-Shared Physical Memory Regions or Type 2 Memory Windows for the RC, RD, and UC Service Types.

Local Invalidate Operations must be posted to the Send Queue. Local Invalidates Operations affect only local HCA memory mapping resources and do not cause any packets to be issued over the link. No resources at the destination QP are affected.

10.7.2.6 FAST REGISTER PHYSICAL MR

The Fast Register Physical MR Operation is allowed on Non-Shared Physical Memory Regions that were created with a Consumer owned key portion of the L_Key, and any associated R_Key, for the RC, RD, and UC Service Types.

When any Service Type QP is created, the Consumer selects whether to enable Fast Register Physical MR Operations or not. If the Consumer enabled Fast Register Physical MR Operations, Fast Register Physical MR Operations must be allowed.

Fast Register Physical MR Operations must be posted to the Send Queue. Fast Register Physical MR Operations affect only local HCA memory mapping resources and do not cause any packets to be issued over the link. No resources at the destination QP are affected.

10.7.3 WORK REQUEST CONTENTS

A Work Request contains all of the information required to perform the requested operation.

The contents of a Work Request for an operation posted to the Send Queue are described in Section 11.4.1.1 Post Send Request on page 612. The contents of a Work Request for an operation posted to the Receive Queue are described in Section 11.4.1.2 Post Receive Request on page 621.

10.7.3.1 SIGNALED COMPLETIONS

Work Requests always generate a Work Completion by default. This is referred to as a Signaled Completion. There is a mechanism where Work Requests posted to the Send Queue may not generate a Work Completion in the associated Completion Queue. This is referred to as an Unsignaled Completion. In order to use Unsignaled Completions, the QP has to

be configured to support Unsignaled Completions and the Work Request must use the Signaling Indicator to request an Unsignaled Completion. Note that if a completion error occurs, a Work Completion will always be generated, even if the signaling indicator requests an Unsignaled Completion.

C10-90: The CI **shall** support both signaled and unsignaled completions.

C10-91: The CI **shall** generate a CQE when a Work Request completed under any of the following conditions:

- The Work Request completed in error.
- The Work Request was submitted to the Receive Queue.
- The Work Request was submitted to a Send Queue configured for only Signaled Completions.
- The Work Request was submitted to a Send Queue configured for Unsignaled Completions but the Work Request requested a Signaled Completion.

C10-92: The CI **shall not** generate a CQE when all of the following conditions have been met for a completed Work Request that was submitted to the Send Queue:

- The Send Queue has been configured to support Unsignaled Completions.
- The Work Request submitted to that Send Queue set the Signaling Indicator to request an Unsignaled Completion
- That Work Request completed successfully.

Work Requests using Unsignaled Completions can be determined to have been completed according to the rules in 10.8.6 Unsignaled Completions.

10.7.3.2 SCATTER/GATHER

A scatter/gather list may contain zero or more Data Segments. The buffers specified in a Work Request scatter/gather list must be registered with the Channel Interface prior to submission. These buffers must be considered to be in the scope of the Channel Interface from the time submitted to a work queue until completion of the Work Request has been confirmed. See 10.8.5 Returning Completed Work Requests and 10.8.6 Unsignaled Completions for a full description on when the completion of a Work Request is confirmed.

C10-93: If the total sum of all of the buffer lengths exceeds the maximum message payload size specified for an RC or UC QP, the CI **shall** report an error.

C10-93.1.1: If the total sum of all the buffer lengths specified for a UD message exceeds the MTU of the port as returned by QueryHCA, the CI **shall not** emit any packets for this message. Further, the CI shall not generate an error due to this condition.

o10-51: If the CI supports RD Service, and if the total sum of all of the buffer lengths exceeds the maximum message payload size specified for an RD QP, the CI **shall** report an error.

A Data Segment is defined by a Virtual Address, L_Key and Length.

C10-94: The CI **shall** support scatter lists for Receive and RDMA Read operations.

C10-95: The CI **shall** support gather lists for Send and RDMA Write operations.

The order in which the Channel Interface accesses the memory described by a scatter/gather list is not defined by the architecture. In particular, this means that after completion of a Work Request whose scatter list contains overlapping Data Segments, the contents of the overlapped memory are undefined.

10.8 WORK REQUEST PROCESSING MODEL

10.8.1 OVERVIEW

The Work Request processing model describes how requests are submitted, processed by the HCA, and the results returned to the Consumer.

10.8.2 SUBMITTING WORK REQUESTS TO A WORK QUEUE

Work Requests are submitted to the HCA through the Verbs abstraction.

Work Queue Elements are abstract. This means that they are not accessible directly by the Consumer of the Channel Interface.

The intent of the architecture is to allow an implementation to pass Work Requests from a User-level Consumer process to the HCA without kernel involvement.

The QP can accept Send Work Requests only when the QP is in states that allow them to be submitted. The rules are as follows:

C10-96: The QP **shall** process Work Requests submitted to the Send Queue as described in the rules that follow:

- Return an immediate error if the QP is in the Reset, Init and RTR states.

- Are processed when the QP is in the RTS state.
- Are completed in error, assuming that processing is able to continue when the QP is in the SQEr or Error state.
- Are enqueued but not processed when the QP is in the SQD state.

For QP's that are not associated with an SRQ, the QP can accept Receive Work Requests only when the QP is in states that allow them to be submitted. The rules are as follows:

C10-97: This compliance statement has been obsoleted.

C10-97.2.1: If the QP is not associated with an SRQ, the QP **shall** process Work Requests submitted to the Receive Queue as described in the rules that follow:

- Return an immediate error if the QP is in the Reset state.
- Are accepted, but incoming messages are not processed when the QP is in the Init state.
- Are processed when incoming messages arrive and the QP is in the RTR, RTS, SQD, or SQEr state.
- Are completed in error, assuming that processing is able to continue when the QP is in the Error state.

Figure 127 shows the transformation of a Work Request into a WQE to be processed by the HCA.

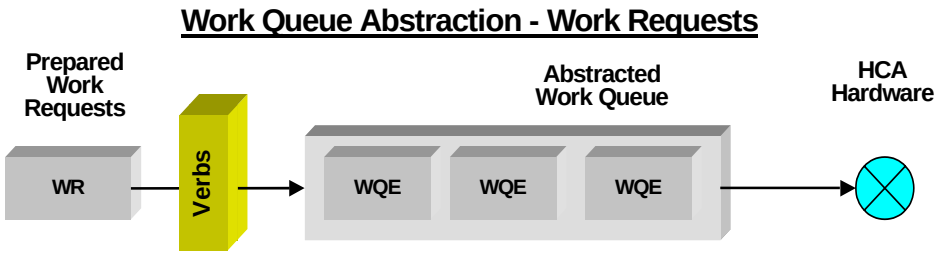


Figure 127 Work Queue Abstraction

For QP's that are associated with an SRQ, the SRQ can accept Receive Work Requests regardless of the SRQ's state. However, if a Consumer's Post Receive Work Request exceeds the SRQ's capacity, an Immediate Error will be returned. The rules are as follows:

o10-51.2.1: If the HCA supports Shared Receive Queues, the SRQ **shall** process Work Requests submitted to it as described in the rules that follow:

- The HCA must return an Async error when an SRQ goes into an error state.
- Are accepted, but incoming messages are not processed (WQE is not retrieved from the SRQ) if the SRQ is not associated with any QP or the SRQ is in the Error state.
- Are processed when incoming messages arrive on a QP that is associated with an SRQ that is not in the error state and that QP is in the RTR, RTS, SQD, or SQEr state.
- Are accepted, but incoming messages are not processed (WQE is not retrieved from the SRQ) if the QP is in the Init, Reset, or Error state.

o10-51.2.2: If the HCA supports Shared Receive Queues and the Consumer posts a Receive WR on a QP that is associated with an SRQ, the CI **shall** return an Immediate Error.

The modifiers in the Work Request are instantiated into the next free WQE in the specified Work Queue and the CI is informed that a new WQE has been added to the queue.

10.8.2.1 SUBMITTING A LIST OF WORK REQUESTS

If the CI supports the Base Queue Management Extensions, the CI must allow:

- a list of Send Queue Work Request to be submitted to the Send Queue,
- a list of Receive Queue Work Request to be submitted to the Receive Queue, and
- if the HCA supports SRQ, a list of Shared Receive Queue Work Requests to be submitted to the SRQ.

The CI must consume one WQE for each WR in the list of WRs. The CI must maintain the same order for the consumed WQEs as the order of the WRs in the list of WRs.

If the list of Work Requests are submitted to a Work Queue is larger than the number of free Work Queue Elements available on the Work Queue, the CI must return an Immediate Error. If an Immediate Error is detected in any Work Request within a list of WRs, the CI must:

- convert all WRs prior to the Immediate Error into WQEs and enqueue the WQEs onto the Work Queue;
- not convert into a WQE the WR that caused the Immediate Error;
- not convert into WQEs, any of the WR that immediately follow the WR that caused the Immediate Error;
- return the first Immediate Error encountered; and

- must return an indication of the Number of WRs that were successfully converted into WQEs and placed on the Work Queue.

10.8.3 WORK REQUEST PROCESSING

Processing of Work Requests submitted to a Work Queue are initiated in the order submitted. There is no ordering between WRs submitted to the send queue and WRs submitted to the receive queue. Send WRs are initiated in the same order they were passed to the Verbs layer with respect to other sends WRs submitted to the same send queue. Likewise, receive WRs are initiated in the same order they were passed to the Verbs layer with respect to other receive WRs submitted to the same receive queue.

Resources associated with a Work Request must be considered to be in the scope of the Channel Interface from the time the Work Request is submitted to a Work Queue until the completion for that Work Request has been confirmed. See 10.8.5 Returning Completed Work Requests and 2.0.4 “Unsignaled Completions” for a description of when a Work Request completion is confirmed.

C10-98: This compliance statement has been obsoleted.

C10-98.2.1: The CI **shall** initiate Send, RDMA Read, RDMA Write and ATOMIC Operations Work Requests in the same order in which they were submitted to a single Send Queue. Work Requests posted to a Send Queue subsequent to a Bind Work Request **shall not** begin execution until the Bind Work Request completes. A Bind Work Request may begin execution before prior Work Requests have completed.

C10-99: This compliance statement has been obsoleted.

C10-99.2.1: For all Service types except RD and Service types that use an SRQ, Work Requests submitted to the same Receive Queue **shall** complete in the same order in which they were submitted.

Work Requests submitted to a single Send Queue complete in the same order as the requests were submitted, according to the Ordering Rules.

The two exception to this rule are: 1) reliable datagrams are permitted to complete out of order on the Receive Queue; and 2) RC and UD QPs that are associated with an SRQ may have Receive WRs completed out of order.

An SRQ can be associated with QPs that may or may not be enabled to use Reserved L_Key. When a WQE is retrieved from an SRQ that refers to memory locations through the Reserved L_Key and the targeted QP does not have Reserved L_Key enabled, the QP shall be put into the error state. The WQE that attempted the Reserved L_Key access will be completed in error, subsequent WQEs pulled from the SRQ onto the same RQ

will be completed with Flushed error, and all other WQEs on the SRQ, including subsequently posted WQEs, are unaffected by the error.

10.8.3.1 RELIABLE DATAGRAM ORDERING RULES

Reliable datagrams originating from a specific Send Queue complete in the same order they were submitted when they are sent to the same Receive Queue.

o10-52: If the CI supports RD Service, Work Requests submitted to the same RD Send Queue **shall** complete in the same order in which they were submitted.

o10-53: If the CI supports RD Service, Work Requests submitted to the same RD Receive Queue **shall** complete in the same order in which they were submitted when the requests originate from the same remote RD Send Queue.

Receive completions from reliable datagrams sent from multiple Send Queues are allowed to be interleaved on the Receive Queue.

10.8.3.2 SHARED RECEIVE QUEUE ORDERING RULES

If the HCA supports Shared Receive Queues, QPs associated with an SRQ, must dequeue WQEs from the SRQ in sequential order.

o10-53.2.1: If the HCA supports Shared Receive Queues, when the HCA receives a message that targets a QP associated with an SRQ, the HCA **shall** retrieve the next available WQE from the SRQ and use it to receive the incoming message.

Multiple QPs that are associated with the same SRQ may dequeue WQEs from the same SRQ. As a result, multiple QPs dequeuing WQEs from a single SRQ destroys any Receive WR posting order that was present.

o10-53.2.2: If the HCA supports Shared Receive Queues:

- Work Requests submitted through the SRQ **shall** be returned as Work Completion through the CQ associated with the QP that received the incoming message;
- for Receive WRs submitted through the SRQ, Receive Work Completions on an RC Service QP **must** be returned in the order the Send WRs were submitted on the remote Send Queue; and
- Successful poll of a WQE **must** guarantee that reposting of a WQE is possible. That is, successful poll of a Work Completion that is associated with a Work Request posted through an SRQ guarantees that posting of another Work Request to the same SRQ is possible.

10.8.3.3 SEND QUEUE ORDERING RULES

As shown in Table 76 Work Request Operation Ordering, ordering semantics for WRs submitted to the Send Queue vary according to the operation type. Some operations can begin processing within the CI while other operations are still outstanding, potentially yielding out-of-order semantics for certain operation sequences. For cases enumerated below, in-order semantics can be guaranteed by setting the Fence Indicator for appropriate WRs. When the Fence Indicator is set for a given WR, that WR cannot begin to be processed until all prior RDMA Read and Atomic operations on the same Send Queue have completed.

C10-100: When the Fence Indicator has been set in a Work Request, the Send Queue **shall not** begin processing that Work Request until all prior RDMA Read and Atomic Operations on that Send Queue have completed.

Here are the cases where the Fence Indicator can be used to guarantee in-order semantics:

- An RDMA Read won't necessarily complete before subsequent Sends, RDMA Writes, or Atomics are initiated and observed by the target. If the target Consumer then modifies memory locations being returned by the RDMA read, the RDMA read could return the newly modified data instead of the original data. Setting the Fence Indicator for the subsequent operation in each case guarantees that the operation will not be observed by the target until all prior RDMA Reads complete.
- An RDMA Read can return data that's been modified by subsequent Sends, RDMA Writes, or Atomics if they target memory locations being returned by the RDMA Read. Setting the Fence Indicator on the subsequent operation in each case guarantees that the operation will not affect data being returned by a prior RDMA read.
- RDMA Read or Atomic operations won't necessarily complete before subsequent Sends, RDMA Writes, or Atomics are initiated and observed by the target. If one of the former operations completes in error on the initiator side because its ACKs fail to return successfully, the subsequent operation could still be observed by the target, and the target Consumer might take some undesired action. Setting the Fence Indicator on the subsequent operation in each case guarantees that it can't be observed by the target unless all prior RDMA Reads and Atomics complete successfully on the initiator side.
- RDMA Read operations may be executed at the target after subsequent Send and Invalidate operation already performed the invalidation at the target. That may cause the RDMA Read operation to fail.

Setting the Fence Indicator on the subsequent operations guarantees that the RDMA Read will fully complete before the invalidation takes place.

The Bind operation has a unique ordering rule: any Work Request posted to a Send Queue subsequent to a Bind must not begin execution until the Bind operation completes. However, note that a Bind operation itself can begin execution in some cases before prior operations have necessarily completed. Also note, a Fast Register PMR or local Relaxed Ordered Invalidate operation submitted after a Bind operation, may be performed before the Bind operation is started.

o10-53.2.3: If the HCA supports the Base Memory Management Extensions, the invalidate operation may take place at any time after it has been posted to the Send Queue, but **must** take place before a completion is generated and before any subsequent WQE has begun execution.

o10-53.2.4: If the HCA supports Local Invalidate Fencing, when the Local Invalidate Fence Indicator has been set in a Local Invalidate Work Request, the Send Queue **shall not** begin processing that Work Request until all prior Work Requests on that Send Queue have completed. Additionally, the Send Queue **shall not** begin processing the Work Request immediately after the Local Invalidate Work Request, until the Local Invalidate Work Request has finished processing.

The Local Invalidate Fence Indicator can be used to guarantee in-order semantics. If the Memory Region referenced by the Local Invalidate is in use at the time the Local Invalidate operation is posted, the HCA may Invalidate the Memory Region. To prevent this from occurring, if the HCA supports fencing on Local Invalidate, the Local Invalidate Fence indicator can be set on the Local Invalidate operation. Setting the Local Invalidate Fence indicator on the Local Invalidate will assure all previous WQEs on the same Send Queue that reference the Memory Region being Invalidated have completed before the Invalidate operation takes place.

Ordering guarantees for processing and completion notifications exist only between Work Requests submitted to the same queue. The ordering across multiple Work Queues is undefined.

C10-101: This compliance statement has been obsoleted.

C10-101.2.1: The CI **shall** provide the guarantees for processing and completion notifications between Work Requests submitted to the same Send Queue as specified by the ordering rules in Table 76 Work Request Operation Ordering.

Ordering Rules:

- Receive Queues are FIFO queues with the exception of the reliable datagram and SRQ issues described above.
- Send Queues are FIFO queues, according to the rules in Table 76 Work Request Operation Ordering. The Fence Indicator can be used to require strict ordering.

Table 76 Work Request Operation Ordering

First Operation \ Second Operation	Send	Bind Window	RDMA Write	RDMA Read	Atomic Op	Fast Register Physical MR	Local Invalidate
Send	#	#	#	#	#	NR	L
Bind Window	#	#	#	#	#	NR	L
RDMA Write	#	#	#	#	#	NR	L
RDMA Read	F	F	F	#	F	NR	L
Atomic Op	F	F	F	#	F	NR	L
Fast Register Physical MR	#	#	#	#	#	#	L
Local Invalidate	#	#	#	#	#	#	#

Table 77 Ordering Rules Key

Symbol	Description
#	Order is always maintained.
NR	Order is not required to be maintained between the Fast Register and the previous operations.
F	Order maintained only if second operation has Fence Indicator set
L	Order maintained only if Invalidate operation has Local Invalidate Fence Indicator set

10.8.4 COMPLETION PROCESSING

The results from a Work Request operation are placed in a Completion Queue Entry (CQE) on the CQ associated with the Work Queue when the request has completed.

A CQE must be generated before a Work Completion can be returned to the Consumer. Note that not all Work Requests will generate a completion, due to unsignaled completions. The rules for when a CQE is generated are outlined in 10.8.5 Returning Completed Work Requests.

C10-102: For completed Work Requests that generate a Work Completion, the CI **shall** place that Work Completion on the CQ associated with the Work Queue.

A CQE is an internal representation of the Work Completion.

10.8.5 RETURNING COMPLETED WORK REQUESTS

All completions are abstracted through the Verbs. The only method of retrieving a Work Completion is through the Verbs.

Except for RD Receive Work Queues and Receive Work Queues associated with an SRQ, Work Completions are always returned in the order submitted to a given Work Queue with respect to other Work Requests on that Work Queue.

For QPs associated with an SRQ, Send Work Completions are always returned in the order submitted to a given work queue with respect to other Work Requests on that work queue. Receive Work Completions may be returned in a different order than the order submitted to the Shared Receive Queue.

Ordering rules of completion entries from multiple work queues associated with a given completion queue are not mandated by this specification.

A retrieved Work Completion is no longer in the domain of the Channel Interface. Therefore, a Work Completion can only be retrieved once.

C10-103: The CI shall not allow a specific Work Completion to be retrieved more than once.

The Work Completion contents are specified in 11.4.2.1 Poll for Completion on page 623.

A Consumer can find out when a Work Completion can be retrieved through polling or notification.

C10-104: The CI **shall** return a Work Completion for a Work Request that completed with a signaled completion.

C10-105: The CI **shall** return a Work Completion for a Work Request submitted to a Send Queue that completed in error.

C10-106: This compliance statement has been obsoleted.

C10-106.2.1: For QPs that are not associated with an SRQ, the CI **shall** return a Work Completion for the completion of a Work Request submitted to a Receive Queue.

o10-53.2.5: If the CI supports SRQ, for QPs that are associated with an SRQ, the CI **shall** return a Work Completion for the completion of a Work Request submitted to a Shared Receive Queue.

o10-53.2.6: If the CI supports SRQ, for a QP that is associated to an SRQ, Work Completions **shall** be returned to the Completion Queue that is associated with the QP's Receive Queue.

C10-107: The CI **shall not** access any buffers associated with the Work Request once the associated Work Completion has been retrieved.

10.8.5.1 FREED RESOURCE COUNT

One of the modifiers returned with the completion is a count that informs the Consumer of the number of work request resources freed by this completion. This applies only to Reliable Datagram Receive Queues. Work request resources refers to Channel Interface resources allocated on behalf of the Consumer, such as available WQEs for a given Work Queue, and not direct Consumer resources, such as buffers.

If this count is zero, this indicates that no receive queue work queue elements have been freed when this Work Completion was generated. If this count is greater than zero, the Consumer can assume that the counter indicates the number of work requests released from the RD RQ. This is useful for the Consumer to keep track of the number of available work requests which can be outstanding.

Buffers associated with the outstanding work request associated with this work completion are no longer considered to be in the scope of the HCA, regardless of the Freed Resource Count.

For most implementations, this count is expected to be one with every work completion.

o10-54: If the CI supports RD Service, when a Work Completion associated with a Work Request posted to an RD RQ is retrieved, the CI **shall** return a count of the number of Work Request resources freed through the Verbs.

10.8.5.2 COMPLETION QUEUE ERRORS

Under certain conditions, a Completion Queue may encounter errors. Two types of CQ errors can occur: the CQ can overrun or it can become inac-

cessible. CQ errors are reported to the Consumer through Affiliated Asynchronous Errors, which are discussed in section 10.10.2.3 Asynchronous Errors on page 531.

CQ overflow occurs when the CQ attempts to generate a CQE and the CQ is already full. Inaccessible CQ is an implementation dependent error where a CQE can not be reported to the CQ, although the CQ is not full. The HCA detects CQ errors and surface them to the CI. When the consumer retrieves completions, the CI only reports good completions or indicates that the CQ is empty.

Following a CQ error, the consumer may still retrieve completions which were outstanding on the CQ at the time the error occurred. After all the outstanding completions have been returned, the CI will indicate that the CQ is empty. Corrupted CQEs are never returned to the consumer as Work Completions.

Following a CQ error, QPs can still be associated, through Create QP, with the CQ that has encountered the error. It is also allowed to modify, query and destroy QPs that are attached to a CQ that has encountered an error. Any attempt to query or resize a CQ that has encountered an error will report an immediate error. Destroy CQ is allowed on such a CQ.

When a CQ encounters an error, in order to be able to use the CQ again, the consumer should:

- Destroy all the QPs that are attached to the CQ
- Destroy the CQ
- Recreate the CQ through the Create Completion Queue verb

If the Consumer requests notification on a CQ that has experienced an Affiliated Asynchronous Error (i.e. a CQ that has overrun or become inaccessible), the CI will not report a completion event.

10.8.6 UNSIGNALED COMPLETIONS

An unsignaled Work Request that completed successfully is confirmed when all of the following rules are met:

- A Work Completion is retrieved from the same CQ that is associated with the Send Queue to which the unsignaled Work Request was submitted.
- That Work Completion corresponds to a subsequent Work Request on the same Send Queue as the unsignaled Work Request.

C10-108: The CI **shall not** access buffers associated with an Unsignaled Work Request once a Work Completion has been retrieved that corresponds to a subsequent Work Request on the same Send Queue.

10.8.7 ASYNCHRONOUS COMPLETION NOTIFICATION

The Consumer may register multiple completion notification routines, one per Completion Event, to be called when a new entry is added to the CQ using the Set Completion Event Handler Verb.

C10-109: This compliance statement has been obsoleted.

C10-109.2.1: A CI shall support registering a single CQ Event Handler per HCA. The CI must ensure that the Consumer can successfully set one completion event handler.

o10-54.2.1: If the CI supports the Base Queue Management Extensions, then the CI must:

- Support one or more Completion Event Handlers;
- Return the number of Completion Event Handlers supported by the HCA through the Query HCA verb;
- Through the Set Completion Event Handler Verb, associate a Completion Handler Address to a Completion Handler Identifier;
- Through the Set Completion Event Handler Verb, clear an existing Completion Event Handler; and
- Through the Create CQ verb, associate the Completion Handler Identifier to a CQ

C10-110: This compliance statement has been obsoleted.

C10-110.2.1: If HCA does not support the Base Queue Management Extensions, the CI shall replace any previous handler associated with a Completion Event identifier with the handler specified in a new Set Completion Event Verb notification.

o10-54.2.2: If HCA supports the Base Queue Management Extensions and the Consumer specifies a non-zero Completion Event Handler Address, the CI shall replace any previous handler associated with a Completion Event identifier with the handler specified in a new Set Completion Event Verb notification. If the Consumer specifies a zero Completion Event Handler Address, the CI shall clear any previous handler.

If the Consumer references an existing Completion Event Handler Identifier and clears a previous handler as described above, then the Consumer cannot use the same Completion Event Handler Identifier until it is returned from a Set Completion Event Handler Verb.

The Request Completion Notification Verb is set on a per-CQ basis. This is a one-shot notification; at most, one notification will be generated per call to this Verb. Once CQ notifications have been enabled, additional Re-

quest Completion Notification calls have no effect. The handler will be called once when the next entry is added to the CQ specified as a modifier to this Verb. The presence of Solicited Events may impact this behavior. See 11.4.2.2 Request Completion Notification on page 627 & 9.2.3 Solicited Event (SE) - 1 bit on page 238 for details.

C10-111: A CQ **shall** have at most one Completion Event notification request outstanding.

C10-112: A CI **shall** generate a single Completion Event when a CQ entry that satisfies the outstanding Completion Event request is added to the CQ.

C10-113: A CI **shall not** generate a Completion Event for existing CQ entries on the specified CQ at the time the completion notification request is registered.

A notification will not be generated until the next entry is added to the CQ.

The following sequence of calls should be used when using Request Completion Notification in order to ensure that a new CQ entry is not missed for the specified CQ.

1) Poll for Completion to dequeue existing CQ entries.
2) Request Completion Notification.
3) Poll for Completion to pick up any CQ entries that were added between the time the first Poll for Completion was called and the notification is enabled.

If a handler has not been registered, a notification will not be generated.

When the handler routine is invoked, an indication of which CQ has generated the completion notification will be supplied. Once the handler routine has been invoked, the Consumer must call Request Completion Notification again to be notified when a new entry is added to the CQ.

C10-114: For each Completion Event, the CI **shall** indicate which CQ caused the generation of that event.

The Consumer is responsible for polling the CQ to retrieve the work completion. This function is not performed automatically when the notification occurs.

10.9 PARTITIONING

This section discusses InfiniBand™ support for partitioning of an InfiniBand™ network. The Verb support for partitioning is contained in the Verbs that perform Queue Pair management, read Channel Interface (CI)

content, and set it. These are documented in 11.2 Transport Resource Management on page 550.

In this discussion, the term “Partition Manager” (PM) refers to the function of the Subnet Manager that deals with partitioning for the CI being discussed; see 13.5 MAD Processing on page 749 for how SMPs are directed to that manager.

10.9.1 INTRODUCTION

Partitioning enforces isolation among systems sharing an InfiniBand™ fabric by requiring that packets contain a 16-bit Partition Key (P_Key) which must match a P_Key stored at the receiver or be discarded; see definition of “match” below (10.9.3 Partition Key Matching). There are no Verbs that directly set the P_Keys sent or matched against in a CA. Verbs instead specify an index into a table of P_Keys: the P_Key_ix, specifying an entry in the P_Key Table. The contents of the P_Key Table are controlled by the subnet’s Partition Manager (PM), which sets them using Subnet Management Packets (SMPs) sent through the subnet’s Subnet Manager.

Subsections appearing below describe the P_Key Table, the matching process, and the way P_Keys are attached to packets. See 14.2.5 Attributes on page 809 for a description of the SMPs which set entries in the P_Key Table.

10.9.1.1 LIMITED AND FULL MEMBERSHIP

A collection of endnodes with the same P_Key in their P_Key Tables are referred to as being *members of a partition*, or *in a partition*. A P_Key Table can specify one of two types of partition membership: Limited or Full. The high-order bit of the partition key is used to record the type of membership in a partition table: 0 for Limited, and 1 for Full. Limited members cannot accept information from other Limited members, but communication is allowed between every other combination of membership types.

10.9.1.2 SPECIAL P_KEYS

There are P_Keys that have special meaning: the default partition key, and the invalid partition keys.

C10-115: The P_Key value 0xFFFF **shall** represent the default partition key.

The default partition key provides Full membership in the default partition.

C10-116: The CI **shall** regard a P_Key as invalid if its low-order 15 bits are all zero.The CI **shall** mark a table entry as invalid by filling it with an invalid P_Key.

C10-117: The PM **must not** use these two P_Key values for any other purposes.

Any P_Key which is not invalid is referred to as valid. The default partition key is valid. A P_Key Table entry containing a valid P_Key is referred to as a valid P_Key Table entry.

10.9.1.3 OPERATION ACROSS SUBNETS

C10-118: Switches or Routers **shall not** modify P_Key values when packets are forwarded/routed within or between subnets.

C10-119: A packet's P_Key **must** match a P_Key stored at the destination CI or CA, or the packet **shall** be discarded; see the definition of "match" below (10.9.3 Partition Key Matching).

In the above case a P_Key sourced in one subnet must be valid in another subnet. Since subnets may have different PMs, this must be arranged to happen, for example by human administration (analogous to assignment of static IP addresses) or by a program dialog between subnets' PMs. The definition of the messages used in such an inter-PM dialog is beyond the scope of this version of the specification.

10.9.2 THE PARTITION KEY TABLE (P_KEY TABLE)

C10-120: Each HCA port and switch SMA port **shall** contain a Partition Key Table (P_Key Table). The valid entries in the P_Key Table **shall** hold P_Keys for all the endnodes with which this CI can communicate.

If a switch or router supports the optional P_Key Enforcement feature, then each of its ports shall contain a Partition Key Table (P_Key Table).

C10-121: The P_Key Table size, meaning the maximum number of entries it can hold, **must** be greater than or equal to one and less than or equal to 65535.

The maximum number of entries that can be held in a P_Key Table can be obtained by using the Query HCA Verb or the NodeInfo SMP. (See 11.2.1.2 Query HCA on page 551 and 14.2.5.3 NodeInfo on page 818.)

C10-122: The CI **must not** provide any interface which allows software above the Verbs to alter the P_Key Table contents or change the validity of any entry in the P_Key Table, except through the use of SMPs.

Verbs allow host software to read entries in the P_Key Table. If the value read is an invalid partition key value, that entry is invalid.

SMPs sent to the endnode are used to read and write entries in the P_Key Table. The operations involved when a table is written are described in a later section.

C10-123: If non-volatile storage is not used to hold P_Key Table contents, then if a PM (Partition Manager) is not present, and prior to PM initialization of the P_Key Table, the P_Key Table **must** act as if it contains a single valid entry, at P_Key_ix = 0, containing the default partition key. All other entries in the P_Key Table **must** be invalid.

10.9.3 PARTITION KEY MATCHING

C10-124: The P_Key field of incoming packets received by an endnode **shall** be matched against a resident P_Key as described in the remainder of this section.

Also see 9.6.1.1.3 BTH:P_Key on page 274 and 18.2.1 Attributes on page 1042.

In the following, let M_P_Key (Message P_Key) be the P_Key in the incoming packet and E_P_Key (Endnode P_Key) be the P_Key it is being compared against in the packet's destination endnode.

- If:
  - neither M_P_Key nor E_P_Key are the invalid P_Key,
  - and the low-order 15 bits of the M_P_Key match the low order 15 bits of the E_P_Key;
  - and the high order bit (membership type) of both the M_P_Key and E_P_Key are not both 0 (i.e., both are not Limited members of the partition)

  then the P_Keys are said to *match*. In this case the incoming packet is accepted and processed normally.
- In all other cases the P_Keys are said to *not match*. The incoming packet must be treated as if it was sent to a nonexistent device, meaning:
  - no ACK is returned
  - optionally, a trap SMP is sent to the SM and a counter is incremented; see 10.9.4 Bad P_Key Trap and P_Key Violations Counter (Optional)
  - there is no other effect on the target endnode.

10.9.4 BAD P_KEY TRAP AND P_KEY VIOLATIONS COUNTER (OPTIONAL)

o10-55: If the CA ports and the GSI port for switches and routers support the trap SMP for P_Key Violations, then if a packet's P_Key does not match, the destination node **shall** send a trap SMP to the SM, specifying

the partitioning class and the Bad P_Key Notification method. The body of the trap SMP **must** contain the offending packet's headers fields as specified in Table 131, “Traps,” on page 812 and Table 138, “Notice DataDetails For Traps 257 and 258,” on page 816. Like all traps, this one **shall** not be sent at a frequency faster than the Subnet Timeout.

o10-56: If the CA ports and the GSI port for switches and routers support the trap SMP for P_Key Violations, then if another P_Key mismatch occurs before the trap can be sent, the data for the new mismatch **shall** replace the previously stored data.

o10-57: If the CA ports and the GSI port for switches and routers support a P_Key Violations counter, then it **shall** have the following characteristics:

- Its minimum size is one bit; its maximum size is 16 bits (unsigned).
- It is incremented whenever the P_Key on a message arriving on a given port does not match (as described in 10.9.3 Partition Key Matching).
- When its value reaches all 1s, further incrementing does not change its value: i.e., it saturates.
- It is initialized by power on reset to zero.

The P_Key Violations counter can be read and set by using a SMP that accesses P_Key Violations component of the PortInfo attribute; see 14.2.5.1 Notices and Traps on page 812.

10.9.5 CI PARTITION SUPPORT

C10-125: Except for the subnet management QP (QP0) and QPs providing RD (Reliable Datagram) or Raw Datagram service, a P_Key **must** be associated with each QP before the QP is used. If a CI has multiple ports, the P_Key Table to which the P_Key index refers **shall** be the P_Key Table of the port that the QP is currently using.

This association is done through Verbs that specify the P_Key_ix of the key to use. If the CI supports automatic path migration, then both the primary and alternate P_Key_ix should map to the same partition key; otherwise APM may not function properly.

C10-126: The CI **shall** attach a QP’s P_Key to all packets sent from the QP’s send queue, except for SMPs, raw datagram packets and packets sent from RD QPs.

SMPs are always sent with the default P_Key, Raw datagram packets do not contain a P_Key, and packets from an RD QP get their P_Keys from the EE context associated with the RD QP.

C10-127: The CI **shall** compare the QP's P_Key to the P_Key contained in all incoming packets, except for raw packets and packets destined for QP0, QP1, and QPs providing RD service. See Section 10.9.8 for rules on P_Key validation in incoming packets on QP0 and QP1.

The comparison is described in 10.9.3 Partition Key Matching. The exceptions to this are described in 10.9.8 Partition Enforcement on Management Queue Pairs and 10.9.5.1 EE Context (Reliable Datagram) Support.

10.9.5.1 EE CONTEXT (RELIABLE DATAGRAM) SUPPORT

o10-58: If the CI supports the RD Service, then it **must** associate a P_Key with each EE Context before the EE Context is used. If a CI has multiple ports, the P_Key Table to which the P_Key index refers **shall** be the P_Key Table of the port that the EE Context is currently using.

o10-59: If the CI supports the RD Service, then the CI **must** attach an EE Context's P_Key to all outgoing Reliable Datagram (RD) packets emitted using that EE Context. All incoming packets using a given EE context **shall** be compared with that EE Context's P_Key as described in 10.9.3 Partition Key Matching.

If the CI supports automatic path migration, then both the primary and alternate P_Key_ix should map to the same partition key; otherwise APM may not function properly.

As stated in that section: if the P_Keys match, the packet is processed normally; otherwise it is silently discarded and, optionally, a trap is issued and the Bad P_Key Counter is incremented as described in that section.

RD service is not used on management queue pairs, so this EE Context support does not apply to them.

10.9.5.2 PARTITION KEY CHANGES

C10-128: When the PM sends a message to a CI port requesting a change to the value of a P_Key Table element, the CI **must** return a response message indicating that the action has either been carried out successfully or not performed for some reason.

C10-129: The CI **shall** guarantee that, after the point in time when it sends a response message to the PM indicating success, the updated P_Key Table values will be used to process all subsequent incoming and outgoing packets traversing the associated port.

This behavior may have begun prior to the PM's receiving the success reply.

10.9.6 TCA PARTITION SUPPORT

C10-130: TCA support for partitioning **must** be the same as that for CIs, with the exception that association of a P_Key with a queue **shall** be done in response to messages that initiate creation of queue pairs, as part of establishing communication with another endnode.

In all other respects, the TCA behaves exactly like a CI in terms of multiple ports, incoming packets, outgoing packets, and changes to the P_Key Table.

10.9.7 FABRIC PARTITION SUPPORT

The switches in the InfiniBand™ fabric may optionally also enforce partitioning. How P_Keys are loaded into switches and how they are used is described in several sections of the chapter describing switches (18.2.4.2.1 Inbound P_Key Enforcement on page 1046 and 18.2.4.4.1 Outbound P_Key Enforcement on page 1054.

10.9.8 PARTITION ENFORCEMENT ON MANAGEMENT QUEUE PAIRS

The two types of management queues each treat partition enforcement in a different way.

C10-131: Packets sent to the Subnet Management Interface QP **shall** always be accepted, regardless of the P_Key contained in the packet.

Isolation and security of management communication are not provided by partitioning, but instead by checking of the Management Key.

Packets sent from a Subnet Management Interface QP may have any P_Key; the default P_Key is used by convention, as described in the management sections.

C10-132: Packets sent to the General Service Interface QP (QP1) **shall** be accepted if the P_Key in the packet matches any valid P_Key in the P_Key Table of the port on which the packet arrived. Matching is defined in 10.9.3 Partition Key Matching.

As stated in that section: if the P_Keys match, the packet is processed normally; otherwise it is silently discarded and, optionally, a trap is issued and the Bad P_Key Counter is incremented as described in that section.

C10-133: Packets sent from the Send Queue of a GSI QP **shall** attach a P_Key associated with that QP, just as a P_Key is associated with non-management QPs.

C10-134: Each switch **shall** also check P_Keys on its GSI QP. Switches **shall** support a P_Key table with at least one entry against which the

P_Key of packets destined for the switch's GSI **shall** be matched, according to the rules as stated in C10-132: above.

10.9.9 RELATED ENFORCEMENT OF MANAGEMENT MESSAGE CHECKING

Checking of the M_Key (see 14.2.4 Management Key on page 806) can optionally be used to prevent anything but an authorized subnet manager from reading any SM data from the SMI, and when the protection test fails, silently discarding the packet that failed. Similarly preventing the writing of SM data through the SMI, with silent discard, is mandatory.

In addition, it is an option to store the M_Key(s), the M_KeyProtectBits which control M_Key checking, and the lease period across power cycles Table 145 PortInfo on page 822.Thus, for example, system initialization techniques cannot assume that a constant default value for that data is present except for first-power-on from the factory.

10.10 ERROR HANDLING SEMANTICS AND MECHANISMS

This section describes the types of errors that are detected at the Channel interface and the response that is generated when those error events occur.

10.10.1 ERROR TYPES

Three classes of errors reported through the Verbs have been defined: immediate errors, completion errors and asynchronous errors. Each of these error classes are described in more detail under their respective headings within 10.10.2 Error Handling Mechanisms. A brief description of each error class follows.

Immediate errors are returned as status from the Verbs.

Completion errors are returned to the Verbs Consumer as status within a Work Completion.

Asynchronous errors are returned through an event handling mechanism.

10.10.2 ERROR HANDLING MECHANISMS

This section describes the mechanisms used to notify the Verb Consumer of errors in the requested operations.

10.10.2.1 IMMEDIATE ERRORS

C10-135: The CI **shall** return Immediate errors upon return of control from the Verb to the Consumer.

The details of these error types are included with each Verb described in the Verbs chapter.

C10-136: If an immediate error is returned from a Verb involved in posting Work Requests to a queue, the CI **shall** ensure that the Work Request has not been posted to the queue.

10.10.2.2 COMPLETION ERRORS

C10-137: A Work Request or WQE that is "completed in error" **shall** have the appropriate completion error returned in the Work Completion status.

The complete list of errors that can be returned in the Work Completion status is described in the Verbs chapter under the Completion Queue Operations (11.4.2.1 Poll for Completion on page 623).

There are two classes of completion errors: Interface checks and processing errors. An interface check is an error in the information supplied to the Channel Interface detected before data is placed onto the link. A processing error is an error encountered during the processing of the work request by the Channel Interface.

10.10.2.3 ASYNCHRONOUS ERRORS

Consumers are notified about asynchronous errors through an asynchronous notification mechanism. In order to be notified when asynchronous errors occur, the Consumer must register a handler using the Set Asynchronous Event Handler Verb.

C10-138: After the asynchronous event handler is registered, all subsequent asynchronous errors **shall** result in a call to the error handler. Asynchronous errors that occur before the error handler is registered **shall** be lost.

The details of these errors are discussed in 11.6.3.2 Affiliated Asynchronous Errors on page 639 and 11.6.3.4 Unaffiliated Asynchronous Errors on page 641.

C10-139: Only one error handler **shall** be registered per HCA. Subsequent calls to the Set Asynchronous Error Handler Verb **shall** cause the previous handler address to be overwritten with the new handler address.

There are two Asynchronous error types:

- Unaffiliated Asynchronous Error. Not related to any specific WQ or CQ.

C10-140: This compliance statement has been deleted.

- Affiliated Asynchronous Error. Related to a specific WQ, CQ or EE context and unable to report the error in a completion. The QP or EE context is transitioned to the Error State.

10.10.3 EFFECTS OF ERRORS ON QP SERVICE TYPES

The different types of IB errors defined have varying effects on queue processing dependent upon the QP's Service Type.

It is important to note that catastrophic errors on the local QP have no direct effect on the remote QP. No attempt is made to send a message below the Verbs to tear down a connection just because a QP has encountered an error.

10.10.3.1 RELIABLE CONNECTION QPS:

C10-141: Immediate errors **shall not** affect QP processing since the Work Request never gets posted to the QP.

C10-142: For Send Queue completion errors, the Work Request on the Send Queue in which the error occurred **shall** be completed in error by the CI. The QP **shall be** placed in the Error State. All subsequent Work Requests **shall** be completed in error

> In the case of local send queue errors, any and all Work Requests on the Send Queue in which the error occurred are completed in error by the Channel Interface. If the local error was an interface check, the remote, corresponding Receive Queue will not consume a Work Request and thus will not surface a completion error. If the local error was a processing error, the remote, corresponding Receive Queue may or may not complete a Work Request in error. The condition of the local and remote memory when a completion error occurs on the send queue for RDMA and atomic operations is specified in 10.3.1.7 Error.

C10-143: For local Receive Queue completion errors, the Work Request on the Receive Queue in which the error occurred **shall** be completed in error by the CI. The QP **shall be** placed in the Error State. All subsequent Work Requests **shall** be completed in error.

C10-144: Affiliated Asynchronous Errors **shall** result in the QP processing being halted such that outstanding Work Requests are not completed successfully by the Channel Interface. The QP **shall** be transitioned to the Error State. Any request in progress on the corresponding Work Queue **shall** be halted and returned with a completion error, unless the CQ has overflowed or become inaccessible. If the CQ has overflowed or become inaccessible, then request in progress **shall** not be returned through the CQ.

C10-145: Table 78 Completion Error Handling for RC Send Queues and Table 79 Completion Error Handling for RC Receive Queues are a more detailed description of the RC error handling actions that **must** be supported by the CI according to the error and Work Queue type.

Descriptions of the error types used in the table are contained in 11.6.2 Completion Return Status on page 634.

Table 78 Completion Error Handling for RC Send Queues

Error Type	Completion Type	Effect on Local QP State	Effect on Remote QP State
Bad Response	Processing	Error	None
Local Length	Interface	Error	None
Local Length	Processing	Error	None
Local QP Operation	Interface	Error	None
Local QP Operation	Processing	Error	None
Local Protection	Interface	Error	None
Local Protection	Processing	Error	None
Memory Mgt. Operation	Interface	Error	None
Remote Invalid Request	Processing	Error	Error
Remote Access	Processing	Error	Error
Remote Operation	Processing	Error	Error
RNR NAK Retry Counter Exceeded	Processing	Error	None
Transport Retry Counter Exceeded	Processing	Error	None
Work Request Flushed	Processing	None	None

Table 79 Completion Error Handling for RC Receive Queues

Error Type	Completion Type	Effect on local QP state	Effect on remote QP state
Local Access	Processing	Error	Error when NAK received
Local Length	Processing	Error	Error when NAK received
Local Protection	Processing	Error	Error when NAK received
Local QP Operation	Processing	Error	Error when NAK received
Memory Mgt Operation	Interface	Error	None[a]
Remote Invalid Request	Processing	Error	Error when NAK received
Work Request Flushed	Processing	None	None

a. In the case of Memory Management Operation Error on the local receive queue, no explicit NAK is sent. Therefore, there is no direct impact on the remote QP. However, the local QP transitions into the Error state as a result of this error, which may cause the remote QP to timeout while executing subsequent WQEs. Therefore, a remote QP timeout error may result when the local QP encounters a Memory Management Operation Error on its Receive Queue.

10.10.3.2 RELIABLE DATAGRAM QPS:

o10-60: If the CI supports RD Service, immediate errors **shall** have no effect on QP/EE processing since the Work Request never gets posted to the QP/EE.

o10-61: If the CI supports RD Service, completion errors on a Send Queue **shall** result in Send Queue processing being halted and the Send Queue state **shall** transition to the Send Queue Error State, as per the state diagram. The Work Request where the error occurred **shall** be completed in error.

In the case of local send queue errors, any and all Work Requests on the Send Queue in which the error occurred are completed in error by the Channel Interface. If the local error was an interface check, the remote, corresponding Receive Queue will not consume a Work Request and thus will not surface a completion error. If the local error was a processing error, the remote, corresponding Receive Queue may or may not complete a Work Request in error. The condition of the local and remote memory when a completion error occurs on the send queue for RDMA and atomic operations is specified in 10.3.1.6 Send Queue Error (SQEr).

o10-62: If the CI supports RD Service, for local Receive Queue completion errors, the Work Request on the Receive Queue in which the error occurred **shall** be completed in error by the CI. All subsequent Work Requests **shall not** be affected by the error.

o10-63: If the CI supports RD Service, completion errors **shall** have no effect on the EE Context State.

o10-64: If the CI supports RD Service, Affiliated Asynchronous Errors **shall** result in the QP processing being halted such that outstanding Work Requests are not completed successfully by the Channel Interface. The QP **shall** transition to the Error State. Any request in progress on the corresponding Work Queue **shall be** halted and returned with a completion error, unless the CQ has overflowed or become inaccessible. If the CQ has overflowed or become inaccessible, then request in progress **shall** not be returned through the CQ.

o10-65: If the CI supports RD Service, when an Affiliated Asynchronous Error is associated only with the QP, the error **shall** have no effect on the EE context. If an Affiliated Error is associated with the EE context, the EE context **shall** transition to the Error state.

o10-66: If the CI supports RD Service, Table 80 Completion Error Handling for RD Send Queues and Table 81 Completion Error Handling for RD Receive Queues are a more detailed description of the RD error handling actions that **must** be supported by the CI for RD:

Table 80 Completion Error Handling for RD Send Queues

Error Type	Completion Type	Effect on Local QP State	Effect on Remote QP State	Effect on Local EE State	Effect on Remote EE State	Error Handling Action
Bad Response	Processing	SQ Error	None	None	None	1
Invalid EE Context Number	Processing	SQ Error	None	None	None	1
Invalid EE Context State	Processing	SQ Error	Indeterminate	EE Error	None	1,3
Local EE Context Operation	Processing	SQ Error	Indeterminate	EE Error	None	1,3
Local Length	Interface	SQ Error	None	None	None	1
Local Length	Processing	SQ Error	None	None	None	1
Local Protection	Interface	SQ Error	None	None	None	1
Local Protection	Processing	SQ Error	Rcv WC Err	None	None	1, 2, 4
Local QP Operation	Interface	SQ Error	None	None	None	1
Local QP Operation	Processing	SQ Error	Rcv WC Err	None	None	1, 2, 4
Local RDD Violation	Processing	SQ Error	None	None	None	1
Memory Mgt Operation	Interface	SQ Error	None	None	None	1
Remote Access	Processing	SQ Error	None	None	None	1
Remote Invalid Request	Processing	SQ Error	None if 1st packet. Opt Rcv WC Err if other than 1st packet.	None	None	1, 2
Remote Invalid RD Request	Processing	SQ Error	None if 1st packet. Opt Rcv WC Err if other than 1st packet.	None	None	1, 2
Remote Operation	Processing	SQ Error	Error	None	None	1,2
RNR NAK Retry Counter Exceeded	Processing	SQ Error	None	None	None	1

Table 80 Completion Error Handling for RD Send Queues (Continued)

Error Type	Completion Type	Effect on Local QP State	Effect on Remote QP State	Effect on Local EE State	Effect on Remote EE State	Error Handling Action
Transport Timeout Retry Counter Exceeded	Processing	SQ Error	None	Error	None	1
Work Request Flushed	Processing	None	None	None	None	None

Table 81 Completion Error Handling for RD Receive Queues

Error Type	Completion Type	Effect on local QP state	Effect on remote QP state	Effect on local EE state	Effect on remote EE state	Error Handling Action
Invalid EE Context State	Processing	Rcv WC Err	SQ Error	EE Error	EE Error	1, 2, 3
Local EE Context Operation	Processing	Rcv WC Err	SQ Error	EE Error	EE Error	1, 2, 3
Local Length	Processing	Rcv WC Err	SQ Error	None	None	1,2
Local Protection	Processing	Rcv WC Err	SQ Error	None	None	1, 2
Local QP Operation	Processing	Rcv WC Err	SQ Error	None	None	1, 2
Remote Aborted	Processing	Rcv WC Err	Indeterminate[a]	None	None	1[b],2
Remote Invalid Request	Processing	None if 1st packet. Opt Rcv WC Err if other than 1st packet.	SQ Error	None	None	1, 2
Work Request Flushed	Processing	None	None	None	None	None

a. May be in SQError or may retry the message later using a different RQ WQE.
b. Action 1 will only happen in the case where the requester abandoned the operation.

Error Handling Actions:

Uninvolved SQs and RQs are unaffected unless they attempt to use an EE-context or QP that is in the error state.

1) The SQ active over the EE-context at the time the error occurred goes to the SQEr state.

   - Receives for the RQ associated with the local SQ placed in SQEr state continue as normal (i.e. are not completed in error, unless they also experience a separate error).
   - Remainder of the WQEs in the SQ which experienced the error are returned in error via Work Completions (WCs).

2) RQ active over the EE-context at the time the error occurred has the WQE which experienced the error returned in error via a WC. The state of the QP associated with the RQ is not affected.
   - All other WQEs in that RQ continue as normal (i.e. are not completed in error, unless they also experience an error).
   - Sends for the SQ associated with the local RQ continue as normal (i.e. are not completed in error, unless they also experience a separate error).
3) Local EE-context is placed in the error state. Remote EE-context state is indeterminate (i.e. may not be in the error state, for example as a result of a source timeout).
   - EE-context cannot be resumed.
   - Must re-establish EE-context.
4) When a Local Protection or Operation SQ Error occurs on RD QPs, the CI on the local side shall emit an InfiniBand no-op (RDMA Write of length 0 with no immediate data) below the Verbs to the RD RQ associated with the local error, assuming the RD channel is still operational. This will cause the in-process RQ Work Request on the remote side to be completed in error. The Receive side cannot depend on receiving that message.

10.10.3.3 UNRELIABLE CONNECTED QPS:

C10-146: Immediate errors **shall** have no effect on QP processing since the Work Request never gets posted to the QP.

C10-147: Completion errors on a Send Queue **shall** result in Send Queue processing being halted and the Send Queue state **shall** transition to the Send Queue Error State, as per the state diagram. The Work Request where the error occurred **shall** be completed in error.

In the case of local send queue errors, any and all Work Requests on the Send Queue in which the error occurred are completed in error by the Channel Interface. The remote, corresponding Receive Queue will not consume a Work Request and thus will not surface a completion error. The condition of the remote memory when a completion error occurs on the send queue for RDMA Write operations is specified in 10.3.1.6 Send Queue Error (SQEr).

C10-148: For local Receive Queue completion errors, the Work Request on the Receive Queue in which the error occurred **shall** be completed in error by the CI. The QP is placed in the Error State. All subsequent Work Requests **shall** be completed in error.

C10-149: Table 82 Completion Error Handling for UC Send Queues and Table 83 Completion Error Handling for UC Receive Queues are a more

detailed description of the UC error handling actions that **must** be supported by the CI according to the error and Work Queue type.

Descriptions of the error types used in the table are contained in 11.6.2 Completion Return Status on page 634.

Table 82 Completion Error Handling for UC Send Queues

Error Type	Completion Type	Effect on Local QP State	Effect on Remote QP State
Local Length	Interface	SQ Error	None
Local QP Operation	Interface	SQ Error	None
Local QP Operation	Processing	SQ Error	None
Local Protection	Interface	SQ Error	None
Local Protection	Processing	SQ Error	None
Memory Mgt Operation	Interface	Error	None
Work Request Flushed	Processing	None	None

Table 83 Completion Error Handling for UC Receive Queues

Error Type	Completion Type	Effect on local QP state	Effect on remote QP state
Local Length	Processing	Error	None
Local Protection	Processing	Error	None
Local QP Operation	Processing	Error	None
Work Request Flushed	Processing	None	None

C10-150: Affiliated Asynchronous Errors **shall** result in the QP processing being halted such that outstanding Work Requests are not completed successfully by the Channel Interface. The QP **shall** transition to the Error State. Any request in progress on the corresponding Work Queue **shall be** halted and returned with a completion error, unless the CQ has overflowed or become inaccessible. If the CQ has overflowed or become inaccessible, then request in progress **shall** not be returned through the CQ.

10.10.3.4 UNRELIABLE DATAGRAM QPS:

C10-151: Immediate errors **shall** have no effect on QP processing since the Work Request never gets posted to the QP.

C10-152: Completion errors on a Send Queue **shall** result in Send Queue processing being halted and the Send Queue state **shall** transition to the

Send Queue Error State, as per the state diagram. The Work Request where the error occurred **shall** be completed in error.

In the case of local send queue errors, any and all Work Requests on the Send Queue in which the error occurred are completed in error by the Channel Interface. The remote, corresponding Receive Queue will not consume a Work Request and thus will not surface a completion error.

C10-153: For local Receive Queue completion errors, the Work Request on the Receive Queue in which the error occurred **shall** be completed in error by the CI. The QP is placed in the Error State. All subsequent Work Requests **shall** be completed in error.

C10-154: Table 84 Completion Error Handling for UD Send Queues and Table 85 Completion Error Handling for UD Receive Queues provide a more detailed description of the UC error handling actions that **must** be supported by the CI according to the error and Work Queue type.

Descriptions of the error types used in the table are contained in 11.6.2 Completion Return Status on page 634.

Table 84 Completion Error Handling for UD Send Queues

Error Type	Completion Type	Effect on Local QP State	Effect on Remote QP State
Local QP Operation	Interface	SQ Error	None
Local QP Operation	Processing	SQ Error	None
Local Protection	Interface	SQ Error	None
Local Protection	Processing	SQ Error	None
Work Request Flushed	Processing	None	None

Table 85 Completion Error Handling for UD Receive Queues

Error Type	Completion Type	Effect on local QP state	Effect on remote QP state
Local Protection	Processing	Error	None
Local QP Operation	Processing	Error	None
Work Request Flushed	Processing	None	None

C10-155: Affiliated Asynchronous Errors **shall** result in the QP processing being halted such that outstanding Work Requests are not completed successfully by the Channel Interface. The QP **shall** transition to the Error State. Any request in progress on the corresponding Work Queue **shall be** halted and returned with a completion error, unless the CQ has over-

flowed or become inaccessible. If the CQ has overflowed or become inaccessible, then request in progress **shall** not be returned through the CQ.

10.10.3.5 RAW QPS:

C10-156: Immediate errors **shall** have no effect on QP processing since the Work Request never gets posted to the QP.

C10-157: Completion errors on a Send Queue **shall** result in Send Queue processing being halted and the Send Queue state **shall** transition to the Send Queue Error State, as per the state diagram. The Work Request where the error occurred **shall** be completed in error.

> In the case of local send queue errors, any and all Work Requests on the Send Queue in which the error occurred are completed in error by the Channel Interface. The remote, corresponding Receive Queue will not consume a Work Request and thus will not surface a completion error.

C10-158: For local Receive Queue completion errors, the Work Request on the Receive Queue in which the error occurred **shall** be completed in error by the CI. The QP is placed in the Error State. All subsequent Work Requests **shall** be completed in error.

C10-159: Table 86 Completion Error Handling for Raw Datagram Send Queues and Table 87 Completion Error Handling for Raw Datagram Receive Queues provide a more detailed description of the UC error handling actions that **must** be supported by the CI according to the error and Work Queue type.

Descriptions of the error types used in the table are contained in 11.6.2 Completion Return Status on page 634.

Table 86 Completion Error Handling for Raw Datagram Send Queues

Error Type	Completion Type	Effect on Local QP State	Effect on Remote QP State
Local Length	Interface	SQ Error	None
Local QP Operation	Interface	SQ Error	None
Local QP Operation	Processing	SQ Error	None
Local Protection	Interface	SQ Error	None
Local Protection	Processing	SQ Error	None
Work Request Flushed	Processing	None	None

Table 87 Completion Error Handling for Raw Datagram Receive Queues

Error Type	Completion Type	Effect on local QP state	Effect on remote QP state
Local Length	Processing	Error	None
Local Protection	Processing	Error	None
Local QP Operation	Processing	Error	None
Work Request Flushed	Processing	None	None

C10-160: Affiliated Asynchronous Errors **shall** result in the QP processing being halted such that outstanding Work Requests are not completed successfully by the Channel Interface. The QP **shall** transition to the Error State. Any request in progress on the corresponding Work Queue **shall be** halted and returned with a completion error, unless the CQ has overflowed or become inaccessible. If the CQ has overflowed or become inaccessible, then request in progress **shall** not be returned through the CQ.

10.10.4 EFFECTS OF TRANSPORT LAYER ERRORS

The different types of errors defined in the Transport Layer 9.9 Error detection and handling on page 396 have varying effects on the Software Transport Interface. Some Transport Layer errors are not visible through the CI. Other Transport Layer errors cause corresponding error(s) to be reported through the Verbs. In general, a single Transport Layer error generates only one corresponding error completion (though other work requests may be flushed) or a single asynchronous error per resource (QP, EEC or CQ) involved in the error.

C10-160.2.1: A requestor CI shall conform to the behavior specified in Table 88: Verbs Level Behavior for Requester Side Errors for all Transport Layer errors specified in Table 56 Requester Side Error Behavior on page 401.

Table 88 Verbs Level Behavior for Requester Side Errors

Transport Error(s)[a]	Service/ Case[b]	Verbs Level Behavior
Packet Sequence Error - Retry Limit Exceeded Implied NAK Sequence Error - Retry Limit Exceeded Local ACK Timeout Error - Retry Limit Exceeded	RC	Transport Retry Counter Exceeded completion QP to Error state other WQEs: Work Request Flushed Error completion
	RD	Transport Retry Counter Exceeded completion QP to SQEr state EEC to Error state[c] other WQEs on Send Queue: Work Request Flushed Error completion

Table 88 Verbs Level Behavior for Requester Side Errors (Continued)

Transport Error(s)[a]	Service/ Case[b]	Verbs Level Behavior
RNR NAK Retry Error - Retry Limit Exceeded	RC	RNR Retry Counter Exceeded completion QP to Error state other WQEs: Work Request Flushed Error completion
	RD	RNR Retry Counter Exceeded completion QP to SQEr state other WQEs on Send Queue: Work Request Flushed Error completion
Unsupported OpCode Unexpected OpCode	RC	Remote Invalid Request Error completion QP to Error state other WQEs: Work Request Flushed Error completion
	RD	Remote Invalid Request Error completion QP to SQEr state other WQEs on Send Queue: Work Request Flushed Error completion
Local Memory Protection Error	RC	Local Protection Error completion QP to Error state other WQEs: Work Request Flushed Error completion
	other	Local Protection Error completion QP to SQEr state other WQEs on Send Queue: Work Request Flushed Error completion
R_Key Violation	RC	Remote Access Error completion QP to Error state other WQEs: Work Request Flushed Error completion
	RD	Remote Access Error completion QP to SQEr state other WQEs on Send Queue: Work Request Flushed Error completion
Remote Operation Error	RC	Remote Operation Error completion QP to Error state other WQEs: Work Request Flushed Error completion
	RD	Remote Operation Error completion QP to SQEr state other WQEs on Send Queue: Work Request Flushed Error completion
Local Operation Error - WQE	RC	Local QP Operation Error completion QP to Error state other WQEs: Work Request Flushed Error completion
	RD	Local QP Operation Error completion OR Local EE Context Operation Error completion QP to SQEr state other WQEs on Send Queue: Work Request Flushed Error completion
	other	Local QP Operation Error completion QP to SQEr state other WQEs on Send Queue: Work Request Flushed Error completion

Table 88 Verbs Level Behavior for Requester Side Errors (Continued)

Transport Error(s)[a]	Service/ Case[b]	Verbs Level Behavior
Local Operation Error - Affiliated or Unaffiliated (only one of these cases will occur)	QP	Local Work Queue Catastrophic Asynchronous Error QP to Error state other WQEs: Work Request Flushed Error completion
	EEC	Local EE Context Catastrophic Asynchronous Error EEC to Error state[c]
	unknown resource	Local Catastrophic Asynchronous Error
Local RDD Violation	RD	Local RDD Violation Error completion QP to SQEr state other WQEs on Send Queue: Work Request Flushed Error completion
Remote RDD Violation Remote Q_Key Violation	RD	Remote Invalid RD Request QP to SQEr state other WQEs on Send Queue: Work Request Flushed Error completion
Length Error	RC	Local Length Error completion QP to Error state other WQEs: Work Request Flushed Error completion
	RD	Local Length Error completion QP to SQEr state other WQEs on Send Queue: Work Request Flushed Error completion
Bad Response	RC	Bad Response Error completion QP to Error state other WQEs: Work Request Flushed Error completion
	RD	Bad Response Error completion QP to SQEr state other WQEs on Send Queue: Work Request Flushed Error completion
CQ Overflow	all	Local Work Queue Catastrophic Asynchronous Error AND CQ Asynchronous Error[d] QP to Error state current and other WQEs: completion can't be reported
All other combinations of Transport Error and Service/Case from Table 56		no effect

a. Transport errors in this table are those cases from section 9.9.2.3 - Table 56 with requester class B, C, D, or F behavior.

b. If a particular transport service type is not listed, then either the error case does not apply to that service type or the error does not manifest any verb level behavior for that service type.

c. When other RD QPs linked to this EEC in the Error state request Send Queue service, those RD QPs will return an Invalid EE Context State completion and transition to the SQEr state.

d. CQ Overflows are to be reported on both the CQ that overflowed as well as all QPs which caused an overflow to occur. When a CQ overflows for the first time, asynchronous errors are generated on both the QP generating the CQE as well as on the overflowing CQ. If the CQ overflows again due to the same QP, no new error is required to be generated. However, if the CQ overflows again due to a different QP (which has previously not caused an overflow), then the Local Work Queue Catastrophic Asynchronous Error must be reported on the new QP.

C10-160.2.2: A responder CI shall conform to the behavior specified in Table 89: Verbs Level Behavior for Responder Side Errors for all Transport Layer errors specified in Table 58 Responder Error Behavior Summary on page 409.

Table 89 Verbs Level Behavior for Responder Side Errors

Transport Error(s)[a]	Service/ Case[b]	Verbs Level
Malformed WQE Local QP Error	all	Local QP Operation Error completion QP to Error state other WQEs: Work Request Flushed Error completion
Unsupported or Reserved Opcode Out of Sequence Opcode - current packet is First or Only Out of Sequence Opcode - current packet is not First or Only	RC	Remote Invalid Request Error completion (if using Receive Queue WQE) OR Invalid Request Local Work Queue Asynchronous Error[c] QP to Error state other WQEs: Work Request Flushed Error completion
Misaligned Atomic Too many RDMA Read or Atomic Requests	RC	Local Access Violation Work Queue Asynchronous Error QP to Error state other WQEs: Work Request Flushed Error completion
Resync Opcode Incomplete WQE	RD	Remote Aborted Error completion (or reset WQE for reuse)
R_Key Violation	RC	Local Access Error completion (if using Receive Queue WQE) OR Local Access Violation Work Queue Asynchronous Error[c] QP to Error state other WQEs: Work Request Flushed Error completion
Length Errors	RC	Remote Invalid Request Error completion (if using Receive Queue WQE) OR Local Access Violation Work Queue Asynchronous Error[c] QP to Error state other WQEs: Work Request Flushed Error completion
	RD	Remote Invalid Request Error completion (if using Receive Queue WQE)
CQ Overflow	all	Local Work Queue Catastrophic Asynchronous Error AND CQ Asynchronous Error[d] QP to Error state current and other WQEs: completion cannot be reported
Local EEC Error	RD	Local EE Context Operation completion (if using Receive Queue WQE) OR Local EE Context Catastrophic Asynchronous Error[c] EEC to Error state[e]
All other combinations of Transport Error and Service/Case from Table 58		no effect

a. Transport errors in this table are those cases from section 9.9.3 - Table 58 with responder class A, C, E, F, G or H behavior.

b. If a particular transport service type is not listed, then either the error case does not apply to that service type or the error does not manifest any verb level behavior for that service type.

c. If a Receive Queue WQE was consumed by this operation, then a completion error is generated. Otherwise, the asynchronous error is generated.
d. CQ Overflows are to be reported on both the CQ that overflowed as well as all QPs which caused an overflow to occur. When a CQ overflows for the first time, asynchronous errors are generated on both the QP generating the CQE as well as on the overflowing CQ. If the CQ overflows again due to the same QP, no new error is required to be generated. However, if the CQ overflows again due to a different QP (which has previously not caused an overflow), then the Local Work Queue Catastrophic Asynchronous Error must be reported on the new QP.
e. When other RD QPs linked to this EEC in the Error state request Send Queue service, those RD QPs will return an Invalid EE Context State completion and transition to the SQEr state.

CHAPTER 11: SOFTWARE TRANSPORT VERBS

11.1 VERBS INTRODUCTION AND OVERVIEW

The Verbs described in this chapter provide an abstract definition of the functionality provided to a host by a host channel interface. Host CIs which are compliant with this specification must exhibit the semantic behavior described by the Verbs.

Since the Verbs define the behavior of the host CI, they may influence the design of software constructs, such as application programming interfaces (APIs), which provide access to the host CI. However, this specification explicitly does not define any such API. In particular, there is no requirement that an API used with a compliant host CI be semantically consistent with the Verbs.

11.1.1 VERB EXTENSIONS

This version of the specification adds the following list of Post-1.1 Verb Extensions to the InfiniBand™ Architecture Specification, Volume 1, Release 1.1:

- Base Queue Management Extensions:
  - enable a list of Work Requests to be submitted to the Send Queue or to the Receive Queue, and
  - enable multiple Completion Event Handlers per HCA.
- Shared Receive Queue:
  - enables multiple RC QPs or UD QPs to reduce the number of receive resources by using a Shared Receive Queue.
- Base Memory Management Extensions:
  - allow the Consumer to allocate Memory Region resources for use in future registrations,
  - allow the Consumer to perform non-pipelined[15] local Memory Region registrations through the Send Queue,
  - allow the Consumer to perform non-pipelined[1] local invalidations of registered Memory Regions and Memory Windows through the Send Queue,
  - allow Consumer to perform remote invalidation of Memory Regions and Memory Windows,

15. For a queue operation, a non-pipelined operation may be performed before the preceding operations posted to the same queue complete.

- enable the direct access of physical addresses by the HCA through the use of a Reserved L_Key,
- provide the following additions to existing memory management verbs:
  - enable Memory Windows to be associated with a single QP,
  - enable the Consumer to own the key portion of the L_Key field for Memory Regions and the key portion of the R_Key for Memory Regions and Memory Windows,
  - explicitly define whether a Memory Region is Shared or Non-Shared.

- Block List based Physical Buffer List, requires Base Memory Management Extensions to also be supported.
- Zero Based Virtual Address (ZBVA), requires Base Memory Management Extensions to also be supported.
- Local Invalidate Fencing, requires Base Memory Management Extensions to also be supported.

o11-0.2.1: If the HCA supports an extension, then it must support all the verbs and modifiers associated with that extension in this specification.

11.1.2 VERB CLASSES

11.1.2.1 MANDATORY VS. OPTIONAL VERBS

Some Verbs are mandatory, and some are required only if an optional feature is supported.

C11-1: A CI **shall** support all Verbs classified as mandatory in Verb Classes.

C11-2: If a CI claims conformance to an optional feature, the CI **shall** support all Verbs associated with that optional feature as indicated in Verb Classes.

11.1.2.2 MANDATORY VS. OPTIONAL VERB FUNCTIONALITY

Some Verbs define functionality that applies only if certain optional features are supported.

C11-3: If a CI supports a given Verb, the CI **shall** support all functionality defined for that Verb that's not indicated as being optional.

C11-4: If a CI supports a given Verb and claims conformance to an optional feature, the CI **shall** support all functionality defined for that Verb that's associated with that optional feature.

11.1.2.3 CONSUMER ACCESSIBILITY

Verb Consumers are the direct users of the Verbs, and are sub-divided into two classes, Privileged and User-Level.

Privileged Consumers are typically those Consumers that operate at a privilege level sufficient to access OS internal data structures directly, and that have the responsibility to control access to the Channel Interface. All Verbs are available for use by Privileged Consumers.

User-Level Consumers are those Consumers that must rely on another agent, having a sufficient high level of privilege, to manipulate OS data structures. Only those Verbs specifically labeled as such are available for use by User-Level Consumers

Table 90 Verb Classes

Verb	Mandatory/Optional Classification	Consumer Accessibility
Open HCA	Mandatory	Privileged
Query HCA	Mandatory	Privileged
Modify HCA Attributes	Mandatory	Privileged
Close HCA	Mandatory	Privileged
Allocate Protection Domain	Mandatory	Privileged
Deallocate Protection Domain	Mandatory	Privileged
Allocate Reliable Datagram Domain	RD Service	Privileged
Deallocate Reliable Datagram Domain	RD Service	Privileged
Create Address Handle	Mandatory	User-Level and Privileged
Modify Address Handle	Mandatory	User-Level and Privileged
Query Address Handle	Mandatory	User-Level and Privileged
Destroy Address Handle	Mandatory	User-Level and Privileged
Create Shared Receive Queue	SRQ	Privileged
Modify Shared Receive Queue	SRQ	Privileged
Query Shared Receive Queue	SRQ	Privileged
Destroy Shared Receive Queue	SRQ	Privileged
Create Queue Pair	Mandatory	Privileged
Modify Queue Pair	Mandatory	Privileged

Table 90 Verb Classes (Continued)

Verb	Mandatory/Optional Classification	Consumer Accessibility
Query Queue Pair	Mandatory	Privileged
Destroy Queue Pair	Mandatory	Privileged
Get Special QP	Mandatory	Privileged
Create Completion Queue	Mandatory	Privileged
Query Completion Queue	Mandatory	Privileged
Resize Completion Queue	Mandatory	Privileged
Destroy Completion Queue	Mandatory	Privileged
Create EE Context	RD Service	Privileged
Modify EE Context Attributes	RD Service	Privileged
Query EE Context	RD Service	Privileged
Destroy EE Context	RD Service	Privileged
Allocate L_Key	Base MM Extensions	Privileged
Register Memory Region	Mandatory	Privileged
Register Physical Memory Region	Mandatory	Privileged
Query Memory Region	Mandatory	Privileged
Deregister Memory Region	Mandatory	Privileged
Reregister Memory Region	Mandatory	Privileged
Reregister Physical Memory Region	Mandatory	Privileged
Register Shared Memory Region	Mandatory	Privileged
Allocate Memory Window	Mandatory	Privileged
Query Memory Window	Mandatory	Privileged
Bind Memory Window	Mandatory	User-Level and Privileged
Deallocate Memory Window	Mandatory	Privileged
Attach QP to Multicast Group	UD Multicast Service	Privileged
Detach QP from Multicast Group	UD Multicast Service	Privileged
Post Send Request	Mandatory	User-Level and Privileged
Post Receive Request	Mandatory	User-Level and Privileged
Poll for Completion	Mandatory	User-Level and Privileged

Table 90 Verb Classes (Continued)

Verb	Mandatory/Optional Classification	Consumer Accessibility
Request Completion Notification	Mandatory	User-Level and Privileged
Set Completion Event Handler	Mandatory	Privileged
Set Asynchronous Event Handler	Mandatory	Privileged

11.2 TRANSPORT RESOURCE MANAGEMENT

11.2.1 HCA

11.2.1.1 OPEN HCA

Description:

Opens the specified HCA and returns an opaque object or handle to uniquely reference each HCA so that Consumers can distinguish between HCAs in the endnode.

C11-5: The handles returned for different HCAs within a system **shall** all be unique.

Once opened, a specific HCA cannot be opened again until after it is closed. Opening the HCA prepares the HCA for use by the Consumer.

C11-6: If Open HCA is called for an HCA that is currently open, the CI shall return the HCA already in use error.

An HCA can be opened in either: block or page mode, but not both modes concurrently. All HCAs are required to support page mode. If an HCA is opened in block mode and if it does not support block mode, the Open HCA must return an immediate error.

Input Modifiers:

- The unique identifier for this HCA. The naming scheme is defined by the OSV.
- The type of Physical Buffer that will be used on the HCA: Block or Page.

Output Modifiers:

- A handle for the HCA instance used as a modifier to other Verbs to specify the desired target HCA.
- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.

- Invalid HCA name.
- HCA already in use.
- Block type Physical Buffers are not supported.

11.2.1.2 QUERY HCA

Description:

Returns the attributes for the specified HCA.

The maximum values defined in this section are guaranteed not-to-exceed values. It is possible for an implementation to allocate some HCA resources from the same space. In that case, the maximum values returned are not guaranteed for all of those resources simultaneously.

Input Modifiers:

- HCA handle.

Output Modifiers:

- The HCA attributes returned are:
  - Vendor specific information such as:
    - Vendor ID.
    - Vendor supplied Part ID.
    - Hardware version.

  These three vendor specific items correspond to the information found in the components NodeInfo:VendorID, NodeInfo:DeviceID and NodeInfo:Revision respectively (see Section 14.2.5.3).

HCA specific values:

- The maximum number of QPs supported by this HCA.
- The maximum number of outstanding work requests on any Work Queue supported by this HCA.
- The maximum number of scatter/gather entries per Work Request supported by this HCA, for all Work Requests other than Reliable Datagram Receive Queue Work Requests.
- The maximum number of scatter/gather entries per Reliable Datagram Receive Queue Work Request supported by this HCA. Zero if RD Service is not supported.
- The maximum number of CQs supported by this HCA.
- The maximum CQE capacity per CQ supported by this HCA.
- The maximum number of Memory Regions supported by this HCA.

- The largest contiguous block that can be registered by this HCA, specified in bytes.
- The maximum number of Protection Domains supported by this HCA.
- The memory page sizes supported by this HCA.
- Ability of this HCA to support Address Handle port number checking.
- Number of physical ports on this HCA.
- Port Attribute list (one list for each port on this HCA):
  - MTU and message size supported for each port of this HCA. The MTU contains the same information found in the PortInfo:MTUCap component for that port (see Section 14.2.5.6).
  - Base LID & LMC for each port of this HCA. These values are valid only when the Port State of the port is Armed or Active. For other port states the values returned are indeterminate. (For more information on the port state see 14.4.5 Port State Transitions on page 877).
  - Contents and length of the Source GID Table. The value of Assigned GIDs are valid only when Port State is Armed or Active. For other states the value of assigned GIDs is indeterminate.
  - PortState of each port of this HCA. see 7.2.7 State Machine Terms on page 169.
  - Contents and length of the partition table. A partition table is required per port. The contents of the partition table are valid only when the Port State is Armed or Active. For other states the contents of the partition table are implementation dependent.
  - The maximum number of virtual lanes supported by this HCA. This value represents the same information as the PortInfo:VLCap component for that port (see Section 14.2.5.6).
  - Optional Bad P_Key counter for each port supported by the HCA.
  - Q_Key Violation counter for each port supported by the HCA.
  - Optional InitTypeReply value (see PortInfo:InitTypeReply component in Section 14.2.5.6). Shall be zero if InitTypeReply is not supported.

- Contents of the Subnet Manager address information for each port of this HCA. This is a table, with entries arranged on a per HCA port basis, which contains the LID and Service Level of the Subnet Manager for that port. If this has not been set by the Subnet Manager (Port State is Armed or Active), this should be set to the permissive LID (0xFFFF).
- Subnet Time Out value for this HCA port. (see 14.2.5.6 PortInfo)
- The following CapabilityMask bits for each port on this HCA as defined in the PortInfo CapabilityMask:
  - IsSM.
  - IsSMDisabled.
  - IsSNMPTunnelingSupported.
  - IsDeviceManagementSupported.
  - IsVendorClassSupported.
  - IsClientReregistrationSupported.

- Maximum number of partitions supported by this HCA. The number of partitions supported must be at least one.
- Node GUID for this HCA.
- Optional System Image GUID. (See Section 14.2.5.3 NodeInfo)
- Indicator that the RNR-NAK generation for RC service is supported.
- The Local CA ACK Delay. This value specifies the maximum expected time interval between the local CA receiving a message and it transmitting the associated ACK or NAK. This is suggested for use in computing the “Local ACK Timeout” field in a CM REQ message, or the “Target ACK Delay” field in a CM REP message. See Local ACK Timeout and Target ACK Delay in Message Field Details. The delay value in microseconds is computed using $4.096\mu s * 2^{(\text{Local CA ACK Delay})}$. The delay value is not a guaranteed upper bound for the CA’s response time, but rather one that can be used as a “maximum expected value” for timeouts.
- Shutdown port capability support indicator.
- InitType support indicator. Indicates InitTypeReply and ability to set InitType value is supported.
- Port Active event support indicator.
- System Image GUID support Indicator.
- Bad P_Key counter support indicator.

- Q_Key Violation counter support indicator.
- Max Responder Resources per QP: The maximum number of RDMA Reads & atomic operations that can be outstanding per QP with this HCA as the target. Shall apply to atomics only if this HCA supports atomic operations.
- Max Responder Resources per EEC: The maximum number of RDMA Reads & atomic operations that can be outstanding per EEC with this HCA as the target. Shall apply to atomics only if this HCA supports RD & atomic operations. For this version of the specification, this value is one.
- Max Responder Resources per HCA: The maximum number of resources used for RDMA Reads & atomic operations by this HCA with this HCA as the target. Shall apply to atomics only if this HCA supports atomic operations.
- Max Initiator Depth per QP: The maximum depth per QP for initiation of RDMA Read & atomic operations by this HCA. Shall apply to atomics only if this HCA supports atomic operations.
- Max Initiator Depth per EEC: The maximum depth per EEC for initiation of RDMA Read & atomic operations by this HCA. Shall apply to atomics only if this HCA supports RD & atomic operations. For this version of the specification, this value is one.
- Ability of this HCA to support atomic operations as well as serialization of atomic operations between itself and other system components such as processors and other HCAs. Three levels of atomicity are defined for this version of the specification:
  - Atomic operations not supported.
  - Atomicity is guaranteed only between QPs on this HCA only.
  - Atomicity is guaranteed between this HCA and any other component, such as CPUs and other HCAs.
- The maximum number of EE contexts that can be supported by this HCA. Shall be zero if the HCA does not support Reliable Datagrams.
- Maximum number of RDDs supported by this HCA. The number of RDDs supported must be at least two. Shall be zero if the HCA does not support Reliable Datagrams.
- The maximum number of Memory Windows supported by this HCA.

- The maximum number of Raw IPv6 Datagram QPs supported by this HCA. Shall be zero if Raw IPv6 Datagrams are not supported.
- The maximum number of Raw Ethertype Datagram QPs supported by this HCA. Shall be zero if Raw Ethertype Datagrams are not supported.
- Ability of this HCA to support modifying the maximum number of outstanding Work Requests per QP.
- Maximum number of multicast groups supported by this HCA. Shall be zero if this HCA does not support IBA unreliable multicast.
- Maximum number of QPs which can be attached to multicast groups for this HCA. Shall be zero if this HCA does not support IBA unreliable multicast.
- Maximum number of QPs per multicast group supported by this HCA. Shall be zero if this HCA does not support IBA unreliable multicast.
- Ability of this HCA to support raw packet multicast.
- Ability of this HCA to support automatic path migration.
- Maximum number of Address Handles supported by this HCA.
- Ability of this HCA to change the primary port for a QP or EEC when transitioning from SQD to SQD state.
- Ability of this HCA to support the Current QP State modifier for Modify Queue Pair.
- The ability of this HCA to support multiple Physical Buffer sizes per Memory Region. This output modifier is returned only if the Base Memory Management Extensions are not supported.
- Ability of this HCA to support the Base Memory Management Extensions. If HCA supports the Base Memory Management Extensions:
  - Value of Reserved L_Key.
  - Maximum Physical Buffer List size supported by this HCA when invoking the Allocate L_Key verb.
  - List of Page sizes supported by this HCA.
  - Bound Type 2 Memory Window Association mechanism:
    - Type 2A - QP Number Association; or
    - Type 2B - QP Number and PD Association.

- Type of Physical Buffer currently in use by this HCA: Page or Block.
- Ability of this HCA to support multiple page sizes per Memory Region.
- Ability of this HCA to support Block List Physical Buffer Lists.
  - Range of Block sizes supported by this HCA.
- Ability of this HCA to support Zero Based Virtual Addresses.
- Ability of this HCA to support Local Invalidate Fencing.
- Ability of this HCA to support the Base Queue Management Extensions. If HCA supports the Base Queue Management Extensions:
  - Maximum number of CQ Event Handlers.
- Ability of this HCA to support the Shared Receive Queues. If HCA supports the Shared Receive Queues:
  - Maximum number of SRQs.
  - Maximum number of WRs per SRQ.
  - Maximum number of Scatter/Gather entries per SRQ WR.
  - Ability to modify the maximum number of WRs per SRQ.

- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete this request.
  - Invalid HCA handle.

11.2.1.3 MODIFY HCA ATTRIBUTES

Description:

Modifies certain attributes in the HCA, as specified by a separate list for each port on the HCA.

If any of the HCA attributes to be modified are invalid, none of the HCA attributes shall be modified. An immediate error shall be returned and the HCA state shall remain unchanged.

Input Modifiers:

- HCA handle.
- Optional System Image GUID.
- Port Attribute list (one list for each port on this HCA):
  - Optional Shutdown Port indicator. If the shutdown port indicator is set, then the port will transition to the Down state. (see PortState in Section 7.2.7)

- Optional InitType value. (see PortInfo:InitType component in Section 14.2.5.6)
- Q_Key Violation counter reset bit. This bit applies only if the HCA supports the Q_Key Violation counter. When this bit is set, the Q_Key Violation counter shall be cleared.
- The following CapabilityMask bits as defined in the PortInfo CapabilityMask:
  - IsSM.
  - IsSNMPTunnelingSupported.
  - IsDeviceManagementSupported.
  - IsVendorClassSupported.

Output Modifiers:

- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid Counter specified.

11.2.1.4 CLOSE HCA

Description:

Closes and resets the specified HCA. This Verb is responsible only for deallocating resources allocated by the Channel Interface to prepare the HCA for use by the Consumer. All other resources are no longer associated or connected with the CI and are the responsibility of the Consumer to handle as deemed necessary.

Input Modifiers:

- HCA handle.

Output Modifiers:

- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.

11.2.1.5 ALLOCATE PROTECTION DOMAIN

Description:

Allocates an unused Protection Domain object. Protection Domain objects are required when creating a Queue Pair, Shared Receive Queue, Address Handle, registering memory and allocating Memory Windows. A Protection Domain object provides an association be-

tween Queue Pairs, Shared Receive Queues, Address Handles, Memory Regions and Memory Windows.

Operations on an unreliable datagram queue pair are allowed only when the Protection Domain of the Queue Pair and the Protection Domain of the Address Handle contained in the work request are identical.

Input Modifiers:

- HCA Handle.

Output Modifiers:

- Protection Domain Object.
- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA handle.

11.2.1.6 DEALLOCATE PROTECTION DOMAIN

Description:

Returns a previously Allocated Protection Domain object for reuse by the Allocate Protection Domain Verb. The Protection Domain object cannot be deallocated if it is still associated with any Queue Pair, Memory Region or Memory Window, or Address Handle.

Input Modifiers:

- HCA Handle.
- Protection Domain object.

Output Modifiers:

- Verb Results:
  - Operation completed successfully.
  - Invalid Protection Domain.
  - Protection Domain is in use.
  - Invalid HCA handle.

11.2.1.7 ALLOCATE RELIABLE DATAGRAM DOMAIN

Description:

Allocates an unused Reliable datagram domain object. Reliable datagram domain objects are required when setting up a reliable datagram Queue Pair and EE contexts. A reliable datagram domain object provides an association between Queue Pairs and EE contexts. Opera-

tions on a reliable datagram queue pair directed at an EE context are allowed only when the reliable datagram domain of the queue pair and the reliable datagram domain of the EE context are identical.

Input Modifiers:

- HCA Handle.

Output Modifiers:

- Reliable datagram domain object.
- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA handle.
  - Reliable Datagrams not supported.

11.2.1.8 DEALLOCATE RELIABLE DATAGRAM DOMAIN

Description:

Returns a previously allocated reliable datagram domain object for reuse by the Allocate Reliable Datagram Domain Verb. The reliable datagram domain object cannot be deallocated if it is still associated with a Queue Pair or an EE context.

Input Modifiers:

- HCA Handle.
- Reliable datagram domain object.

Output Modifiers:

- Verb Results:
  - Operation completed successfully.
  - Invalid reliable datagram domain.
  - Reliable datagram domain is in use.
  - Invalid HCA handle.
  - Reliable Datagrams not supported.

11.2.2 ADDRESS MANAGEMENT VERBS

These Verbs create, manipulate and destroy address handles. These address handles are only used for Work Requests submitted to Unreliable Datagram Service Type QPs.

11.2.2.1 CREATE ADDRESS HANDLE

Description:

The purpose of the Create Address Handle Verb is to create an address handle for the address vector passed in through the Verbs. The normal completion for this Verb returns the address handle. The address handle is used to reference a local or global destination in all UD QP Post Sends.

Input Modifiers:

- HCA Handle.
- Protection domain
- Address vector, for UD transports only, containing:
  - Service level.
  - Send Global Routing Header Flag.
  - Destination LID. If destination is in same subnet, LID = final destination; otherwise LID = router LID. DLID = 0xFFFF is illegal except when the destination QP is QP0.
  - For global destination:
    - Flow label.
    - Hop limit.
    - Traffic class.
    - Source GID index.
  - For global destination or Multicast address:
    - Destination's GID or MGID.
  - Maximum Static Rate.
  - Source Path Bits. When the DLID = 0xFFFF, the processing described in Section 14.2.2.1 applies, which may result in the Source Path Bits being ignored.
  - Physical Port

Output Modifiers:

- Address Handle.
- Verb results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid protection domain
  - Insufficient resources to complete request.
  - Invalid Port

11.2.2.2 MODIFY ADDRESS HANDLE

Description:

The purpose of the Modify Address Handle Verb is to change an address vector associated with the address handle passed in by the Consumer.

If any of the Address Handle attributes to be modified are invalid, none of the Address Handle attributes shall be modified. An immediate error shall be returned and the Address Handle state shall remain unchanged.

Input Modifiers:

- HCA Handle.
- Address Handle.
- Address vector, for UD transports only, containing:
  - Service level.
  - Send Global Routing Header Flag.
  - Destination LID. If destination is in same subnet, LID = final destination; otherwise LID = router LID. DLID = 0xFFFF is illegal except when the destination QP is QP0.
  - For global destination:
    - Flow label.
    - Hop limit.
    - Traffic class.
    - Source GID index.
  - For global destination or Multicast address:
    - Destination's GID or MGID.
  - Maximum Static Rate.
  - Source Path Bits. When the DLID = 0xFFFF, the processing described in Section 14.2.2.1 applies, which may result in the Source Path Bits being ignored.
  - Physical Port

Output Modifiers:

- Verb results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid address handle.
  - Invalid Port

11.2.2.3 QUERY ADDRESS HANDLE

Description:

The purpose of the Query Address Handle Verb is to obtain the address vector associated with the address handle passed in by the Consumer.

Input Modifiers:

- HCA Handle.
- Address Handle.

Output Modifiers:

- Address vector, for UD transports only, containing:
  - Service level.
  - Send Global Routing Header Flag.
  - Destination LID. If destination is in same subnet, LID = final destination; otherwise LID = router LID.
  - For global destination:
    - Flow label.
    - Hop limit.
    - Traffic class.
    - Source GID index.
  - For global destination or Multicast address:
    - Destination's GID or MGID.
  - Maximum Static Rate.
  - Source Path Bits.
- Protection domain
- Physical Port
- Verb results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid address handle.

11.2.2.4 DESTROY ADDRESS HANDLE

Description:

The purpose of the Destroy Address Handle Verb is to remove an address vector and its associated address handle from the CI. After the address handle is removed, it can no longer be used to reference the destination.

Input Modifiers:

- HCA Handle.
- Address Handle.

Output Modifiers:

- Verb results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid address handle.

11.2.3 SHARED RECEIVE QUEUE

11.2.3.1 CREATE SHARED RECEIVE QUEUE

Description:

Creates an SRQ for the specified HCA.

A set of initial SRQ attributes must be specified by the Consumer.

o11-0.2.2: If the CI supports SRQ and any of the required initial attributes are illegal or missing, an error shall be returned and the SRQ shall not be created.

On success, a handle to the newly created SRQ is returned.

Input Modifiers:

- HCA handle.
- Protection Domain.
- The maximum number of outstanding Work Requests the Consumer expects to submit to the Shared Receive Queue.
- The maximum number of scatter elements the Consumer will specify in a Work Request submitted to the Shared Receive Queue.

Output Modifiers:

- The SRQ handle for the newly created SRQ.
- The actual number of outstanding Work Requests supported on the Shared Receive Queue. If an error is not returned, this is guaranteed to be greater than or equal to the number requested. (This may require the Consumer to increase the size of the CQ.)
- The actual number of scatter elements that can be specified in Work Requests submitted to the Shared Receive Queue. If an error is not returned, this is guaranteed to be greater than or equal to the number requested.
- Verb Results:

- Operation completed successfully.
- Insufficient resources to complete request.
- Invalid HCA handle.
- Maximum number of Work Requests requested exceeds HCA capability.
- Maximum number of scatter elements requested exceeds HCA capability.
- Invalid Protection Domain.
- HCA does not support SRQ.

11.2.3.2 QUERY SHARED RECEIVE QUEUE

Description:

Returns the attributes of the specified SRQ.

Input Modifiers:

- HCA handle.
- SRQ Handle.

Output Modifiers:

- Protection Domain.
- The actual number of outstanding Work Requests supported on the Shared Receive Queue.
- The actual number of scatter elements that can be specified in Work Requests submitted to the Shared Receive Queue.
- SRQ Limit. If the SRQ Limit is armed, returns the current SRQ Limit. If the SRQ is not armed, returns zero.
- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid SRQ handle.
  - SRQ is in the Error State.
  - HCA does not support SRQ.

11.2.3.3 MODIFY SHARED RECEIVE QUEUE

Description:

Modifies the attributes of an SRQ for the specified HCA.

If any of the modify attributes are invalid, none of the attributes shall be modified.

Input Modifiers:

- HCA handle.
- SRQ handle.
- The SRQ attributes to modify and their new values. The SRQ attributes that can be modified after the SRQ has been created are:
  - The maximum number of outstanding Work Requests the Consumer expects to submit to the Shared Receive Queue, if resizing of the SRQ is supported by the HCA.
  - SRQ Limit. If the SRQ Limit is greater than zero, then it shall be armed upon returning from this verb.

Output Modifiers:

- The actual number of outstanding Work Requests supported on the Shared Receive Queue. If an error is not returned, this is guaranteed to be greater than or equal to the number requested. (This may require the Consumer to increase the size of the CQ.)
- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA handle.
  - Invalid SRQ handle.
  - SRQ is in the Error State.
  - HCA does not support resizing SRQ.
  - Maximum number of Work Requests requested exceeds HCA capability.
  - SRQ Limit exceeds maximum number of Work Requests allowed on the SRQ.
  - More outstanding entries on WQ than size specified.
  - HCA does not support SRQ.

11.2.3.4 DESTROY SHARED RECEIVE QUEUE

Description:

Destroys an SRQ for the specified HCA.

Input Modifiers:

- HCA handle.
- SRQ handle.

Output Modifiers:

- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid SRQ handle.
  - SRQ still has QPs associated with it.
  - HCA does not support SRQ.

11.2.4 QUEUE PAIR

11.2.4.1 CREATE QUEUE PAIR

Description:

Creates a QP for the specified HCA.

A set of initial QP attributes must be specified by the Consumer.

C11-7: If any of the required initial attributes are illegal or missing, an error **shall** be returned and the Queue Pair **shall not** be created.

On success, a handle to the newly created QP and the QP number are returned.

o11-0.2.3: If an HCA supports SRQ, the CI must allow UD and RC QPs to be associated with an SRQ.

Input Modifiers:

- HCA handle.
- SRQ Handle, if QP is to be associated to an SRQ.
- The QP attributes that must be specified at QP create time are:
  - The CQ to be associated with the Send Queue.
  - The CQ to be associated with the Receive Queue.
  - The maximum number of outstanding Work Requests the Consumer expects to submit to the Send Queue.
  - The maximum number of outstanding Work Requests the Consumer expects to submit to the Receive Queue. This value is ignored if the QP is associated with an SRQ.
  - The maximum number of scatter/gather elements the Consumer will specify in a Work Request submitted to the Send Queue.
  - The maximum number of scatter/gather elements the Consumer will specify in a Work Request submitted to the Receive Queue. This value is ignored if the QP is associated with an SRQ.

- Reliable datagram domain to be associated with this QP. Applicable only to RD QPs.
- The Signaling Type must be specified for the Send Queue on this QP. The valid types are:
  - Non-selectable: All Work Requests submitted to the Send Queue always generate a completion entry.
  - Selectable: Consumer must specify on each Work Request submitted to the Send Queue whether to generate a completion entry for successful completions.
- The Consumer must specify a Protection Domain.
- The Transport Service Type requested for this QP. Valid Service Types are:
  - Reliable Connection.
  - Reliable Datagram.
  - Unreliable Connection.
  - Unreliable Datagram.
- Enable or disable Fast Register PMR and Reserved L_Key operations.

Output Modifiers:

- The handle for the newly created QP.
- QP number.
- The actual number of outstanding Work Requests supported on the Send Queue. If an error is not returned, this is guaranteed to be greater than or equal to the number requested. (This may require the Consumer to increase the size of the CQ.)
- The actual number of outstanding Work Requests supported on the Receive Queue. If an error is not returned, this is guaranteed to be greater than or equal to the number requested. (This may require the Consumer to increase the size of the CQ.) This value is not returned if an SRQ is associated with the QP.
- The actual number of scatter/gather elements that can be specified in Work Requests submitted to the Send Queue. If an error is not returned, this is guaranteed to be greater than or equal to the number requested.
- The actual number of scatter/gather elements that can be specified in Work Requests submitted to the Receive Queue. If an error is not returned, this is guaranteed to be greater than or equal to the number requested. This value is not returned if an SRQ is associated with the QP.
- Verb Results:

- Operation completed successfully.
- Insufficient resources to complete request.
- Invalid HCA handle.
- Invalid CQ handle.
- Maximum number of Work Requests requested exceeds HCA capability.
- Maximum number of scatter/gather elements requested exceeds HCA capability.
- Invalid Protection Domain.
- Invalid Service Type for this QP.
- Invalid Reliable Datagram Domain.
- Invalid SRQ handle.
- HCA doesn't support Base Memory Management Extensions.
- HCA does not support SRQ.

11.2.4.2 MODIFY QUEUE PAIR

Description:

C11-8: Upon invocation of this Verb, the CI **shall** modify the attributes for the specified QP and then **shall** cause the QP to transition to the specified QP state.

Only a subset of the QP attributes can be modified in each of the QP states.

C11-9: If any of the QP attributes to be modified are invalid or the requested state transition is invalid, none of the QP attributes **shall** be modified. An immediate error **shall** be returned and the QP state **shall** remain unchanged.

Some QP attributes can be modified with outstanding Work Requests. WRs can be outstanding on the Receive Queue when the QP is in the Init, RTR, RTS, SQD & SQEr state and on the Send Queue when the QP is in the RTS & SQD state. Any outstanding Work Request on a Work Queue may not execute as expected if the QP modifiers are changed. For instance, if RDMA Reads, which were successfully posted, are outstanding when the QP is modified to no longer allow RDMA Reads, some outstanding in-flight RDMA Reads may complete while pending WRs may fail.

C11-10: The properties and requirements of the QP state transitions **shall** be supported as shown in Table 91.

Table 91 QP State Transition Properties

Transition	Required Attributes	Optional Attributes	Actions
Reset to Init	Enable/disable RDMA[a] and Atomic Operations. P_Key index. Primary Physical port. Q_Key for unconnected Service Types.	None.	Enable posting to the Receive Queue.
Init to Init	None.	Enable/disable RDMA[a] and Atomic Operations. P_Key index. Primary Physical port. Q_Key for unconnected Service Types.	No transition.
Init to RTR	Remote Node Address Vector (Connected QPs only). Destination QP Number (RC/UC QPs only). RQ PSN. Number of responder resources for RDMA Read/atomic ops. Minimum RNR NAK Timer Field (RC and RD QPs only).	Alternate path address information (RC/UC QPs only). Enable/disable RDMA[a] and Atomic Operations. P_Key index. Q_Key. Number of WQEs.	Activate receive processing.
RTR to RTS	Local ACK Timeout (RC QP only) Retry count (RC QP only) RNR retry count (RC QP only) SQ PSN. Number of Outstanding RDMA Read/atomic ops at destination.	Enable/disable RDMA[a] & Atomic Operations. Q_Key. Alternate path address information (RC/UC QPs only). Path migration state. Number of WQEs. Current QP State. Minimum RNR NAK Timer Field (RC and RD QPs only).	Activate send processing.

Table 91 QP State Transition Properties (Continued)

Transition	Required Attributes	Optional Attributes	Actions
RTS to RTS (no transition)	None.	Enable/Disable RDMA and Atomic Operations.[a] Q_Key. Alternate path address information (RC/UC QPs only). Path Migration state. Number of WQEs. Current QP State. Minimum RNR NAK Timer Field (RC and RD QPs only).	No transition.
SQEr to RTS	None.	Enable/disable RDMA & Atomic Operations.[a] Q_Key. Number of WQEs. Current QP State. Minimum RNR NAK Timer Field (RD QPs only)	Activate send processing.
RTS to SQD	None.	None.	Deactivate send processing.

Table 91 QP State Transition Properties (Continued)

Transition	Required Attributes	Optional Attributes	Actions
SQD to SQD	None.	Enable/Disable RDMA & Atomic Operations. [a] Remote Node Address Vector (Connected QPs only).[b] [c] Alternate path address information (RC/UC QPs only). Path migration state. Number of Outstanding RDMA Read/atomic ops at destination. [b] Number of local RDMA Read/atomic responder resources. [b] Q_Key. P_Key index. [b] Local ACK Timeout (RC QP only). [b] Retry count (RC QP only). [b] RNR retry count (RC QP only). [b] Number of WQEs. Primary physical port associated with QP if HCA supports the capability to change the primary physical port for a QP when transitioning from SQD to SQD state (RC QPs only).[b] Minimum RNR NAK Timer Field (RC and RD QPs only).	Modify QP attributes

Table 91 QP State Transition Properties (Continued)

Transition	Required Attributes	Optional Attributes	Actions
SQD to RTS	None.	Enable/Disable RDMA and Atomic Operations.[a] Q_Key. Alternate path address information (RC/UC QPs only). Path migration state. Number of WQEs. Current QP State. Minimum RNR NAK Timer Field (RC and RD QPs only).	Activate send processing.
Any State to Error	None.	None allowed.	Queue processing is stopped. Work Requests pending or in process are completed in error, when possible.
Any state to Reset	None.	None allowed.	QP attributes are reset to the same values after the QP was created. Outstanding Work Requests are removed from the queues without notifying the Consumer.

a. If disable RDMA is requested while incoming RDMAs to that queue are in process, it is indeterminate when the disable will take effect. It is up to the Consumer to coordinate the disable with the remote QPs.
b. It is allowed to change this attribute only when the SQ is drained.
c. When changing the Remote Node Address Vector, the Path MTU cannot be changed in the SQD2SQD transition.

Input Modifiers:

- HCA handle.
- QP handle.
- The QP attributes to modify and their new values. The QP attributes that can be modified after the QP has been created are:
  - Next QP state. If specify the current state, only the QP attributes will be modified.
    - Enable or disable Send Queue Drained, Asynchronous Affiliated Event Notification. This modifier is only applicable when the next QP state chosen is SQD.

- Primary P_Key index. Not applicable on a Raw Datagram or Reliable Datagram QPs.
- The Q_Key that incoming Datagram messages are checked against and possibly used as the outgoing Q_Key (based on the WR Q_Key). This applies only to UD & RD QPs.
- PSN for Send QP. Applicable only for RC, UD & UC QPs.
- The maximum number of outstanding Work Requests the Consumer expects to submit to the Send Queue, if resizing of the work queues is supported by the HCA.
- The maximum number of outstanding Work Requests the Consumer expects to submit to the Receive Queue, if resizing of the work queues is supported by this HCA. This value is ignored if the QP is associated with an SRQ.

The following attributes are not applicable if the QP specified is a Special QP: SMI QP (QP0), GSI QP (QP1), Raw IPv6 and Raw Ethertype.

- Primary physical port associated with this QP. Not applicable on RD QPs. Applicable for the SQD to RTS transition on RC QPs, if supported by the HCA. Applicable for the Init to Init transition on all QPs.
- PSN for Receive QP. Applicable only for RC & UC QPs.
- Enable or disable incoming RDMA Reads on this QP. Not applicable on Unreliable Service Type QPs.
- Enable or disable incoming RDMA Writes on this QP. Not applicable on UD Service Type QPs.
- Enable or disable incoming Atomic Operations on this QP. Not applicable on Unreliable Service Type QPs.
- Destination QP number. Applicable only to RC & UC QPs.
- Initiator Depth: Number of RDMA Reads & atomic operations outstanding at any time. Applicable only to RC QPs.
- Responder Resources: Number of responder resources for handling incoming RDMA Reads & atomic operations. This value may be rounded up to a supported value, not to exceed the maximum value allowable for QPs for this HCA. Applicable only to RC QPs.
- Minimum RNR NAK Timer Field Value. When a message arrives which is targeted at a local receive queue, and that receive queue has no receive work requests outstanding, the CI may respond to the initiator with an RNR NAK packet. This modifier is the minimum value which shall be sent in the Timer Field of such an RNR NAK packet; it does not affect RNR NAKs sent for other reasons. If the value specified is not one

of the RNR NAK Timer Field values defined in Table 45 Encoding for RNR NAK Timer Field on page 330, the CI shall return an immediate error. Applicable only to RC and RD QPs.

- Current QP State. This modifier is only valid when moving to the RTS state.
- Address vector, for RC & UC transports only, containing:
  - Service level.
  - Send Global Routing Header Flag.
  - Destination LID. If destination is in same subnet, LID = final destination; otherwise LID = router LID.
  - Path MTU.
  - Maximum Static Rate.
  - Local ACK Timeout (see Sections 12.7.33 Target ACK Delay on page 678 and 12.7.34 Local ACK Timeout on page 678). Applicable only to RC QPs.
  - Retry count. Applicable only to RC QPs.
  - RNR retry count. Applicable only to RC QPs.
  - Source Path Bits.
  - For global destination:
    - Traffic class.
    - Flow label.
    - Hop limit.
    - Source GID index.
    - Destination's GID.
- Alternate path address information, applicable only for RC & UC QPs when this CI support automatic path migration. Note: the path MTU for the alternate path must be the same as for the primary path. The specifics are:
  - Alternate path P_Key index. The alternate path P_Key index should map to the same partition key as the primary P_Key index; otherwise APM may not function properly
  - Alternate path Primary Physical port.
  - Alternate path address vector, containing:
    - Service level.
    - Send Global Routing Header Flag.
    - Destination LID. If the destination is in the same subnet, LID = final destination; otherwise LID = router LID.
    - Maximum Static Rate.

- Local ACK Timeout (see Sections 12.7.33 Target ACK Delay on page 678 and 12.7.34 Local ACK Timeout on page 678). Applicable only to RC QPs.
- Source Path Bits.

For global destination:

- Traffic class.
- Flow label.
- Hop limit.
- Source GID index.
- Destination's GID.

- Path Migration state. Valid only if this HCA supports automatic path migration. Valid states to set are:
  - Migrated.
  - Rearm.

Output Modifiers:

- The actual number of outstanding Work Requests supported on the Send Queue, if resizing of the work queues is supported by the HCA. If an error is not returned, this is guaranteed to be greater than or equal to the number requested. (This may require the Consumer to increase the size of the CQ.)
- The actual number of outstanding Work Requests supported on the Receive Queue. If an error is not returned, this is guaranteed to be greater than or equal to the number requested. (This may require the Consumer to increase the size of the CQ.) This value is not returned if an SRQ is associated with the QP.
- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA handle.
  - Invalid QP handle.
  - Cannot change QP attribute.
  - Atomic operations not supported.
  - P_Key index out of range.
  - P_Key index specifies invalid entry in P_Key table.
  - Invalid QP state.
  - Invalid path migration state.
  - MTUCap of HCA port exceeded.

- Invalid Port.
- Invalid Service Type for this QP
- Maximum number of Work Requests requested exceeds HCA capability.
- Invalid RNR NAK Timer Field value.
- More outstanding entries on WQ than size specified.
- QP still has Type 2A MWs bound to it.

11.2.4.3 QUERY QUEUE PAIR

Description:

Returns the attribute list and current values for the specified QP. This QP handle can be any QP handle supplied by the Verbs.

Input Modifiers:

- HCA handle.
- QP handle.

Output Modifiers:

- The QP attributes. The list of attributes returned by the query are:
  - The QP number.
  - SRQ Handle, returned if QP is associated with an SRQ.
  - Handle of the Completion Queue associated with the Send Queue.
  - Handle of the Completion Queue associated with the Receive Queue.
  - The actual number of outstanding requests supported on the Send Queue.
  - The actual number of outstanding Work Requests supported on the Receive Queue. This value is not returned if an SRQ is associated with the QP.
  - The actual number of scatter/gather entries supported on Work Requests submitted to the Send Queue.
  - The actual number of scatter/gather elements that can be specified in Work Requests submitted to the Receive Queue. This value is not returned if an SRQ is associated with the QP.
  - Fast Register PMR and Reserved L_Key operations enabled or disabled.
  - Current QP State. Where the state is one of the following:
    - Reset

- Initialized
- Ready to Receive (RTR)
- Ready to Send (RTS)
- SQ Error
- SQ Drain (SQD)
  The following modifiers are valid only when the QP is in the SQD state:
  - Send Queue Draining
  - Send Queue Drained
- Error

The following attributes are not defined if the QP is in the Reset state.

- PSNs for Send & Receive QPs. Applicable only for RC & UC QPs.
- RDMA Read enable.
- RDMA Write enable.
- Atomic Operation enable.
- Primary physical port associated with this QP. Not applicable on RD QPs.
- Primary P_Key index. Not applicable for RD & Raw Datagram QPs.
- Q_Key for the Receive Queue. Not applicable to RC, UC & Raw Datagram QPs.
- Reliable Datagram Domain. Applicable only to RD QPs.
- Destination QP number. Applicable only to RC & UC QPs.
- Initiator Depth: Number of RDMA Reads & Atomic Operations outstanding at any time on the destination QP. Applicable only to RC QPs.
- Responder Resources: Number of responder resources for handling incoming RDMA Reads & atomic operations. Applicable only to RC QPs.
- Minimum RNR NAK Timer Field Value. When a message arrives which is targeted at a local receive queue, and that receive queue has no receive work requests outstanding, the CI may respond to the initiator with an RNR NAK packet. This modifier is the minimum value which shall be sent in the Timer Field of such an RNR NAK packet; it does not affect RNR NAKs sent for other reasons. Applicable only to RC and RD QPs.

- Primary Address vector, for RC & UC transports only, containing:
  - Service level.
  - Send Global Routing Header Flag.
  - Destination LID. If destination is in same subnet, LID = final destination; otherwise LID = router LID.
  - Path MTU.
  - Maximum Static Rate.
  - Local ACK Timeout (see Sections 12.7.33 Target ACK Delay on page 678 and 12.7.34 Local ACK Timeout on page 678). Applicable only to RC QPs.
  - Retry count. Applicable only to RC QPs.
  - RNR retry count. Applicable only to RC QPs.
  - Source Path Bits.
  - For global destination:
    - Traffic class.
    - Flow label.
    - Hop limit.
    - Source GID index.
    - Destination's GID.
- Alternate path address information, returned only for RC & UC QPs. Valid only when automatic path migration is enabled.
  - Alternate path P_Key index.
  - Alternate physical port.

  Path address vector, containing:
  - Service level.
  - Send Global Routing Header Flag.
  - Destination LID. If the destination is in the same subnet, LID = final destination; otherwise LID = router LID.
  - Maximum Static Rate.
  - Local ACK Timeout (see Sections 12.7.33 Target ACK Delay on page 678 and 12.7.34 Local ACK Timeout on page 678). Applicable only to RC QPs.
  - Source Path Bits.

    For global destination:
    - Traffic class.

- Flow label.
- Hop limit.
- Source GID index.
- Destination's GID.

- Path migration state. Valid only if this HCA supports automatic path migration.

- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid QP handle.

11.2.4.4 DESTROY QUEUE PAIR

Description:

Destroys the specified QP.

C11-11: Any resources allocated by the Channel Interface in order to process Work Requests on the QP **must** be deallocated as part of the destroy operation.

A QP instance is allowed to have Work Requests outstanding when a request to destroy the QP is made. When a QP is destroyed, any outstanding Work Requests are no longer considered to be in the scope of the Channel Interface. It is the responsibility of the Consumer to clean up resources associated with a Work Request.

C11-12: Incoming operations destined for a QP that has been destroyed **shall be** discarded.

The CI does not guarantee that CQEs generated for a QP prior to its destruction can be retrieved from the CQ after that QP has been destroyed.

Input Modifiers:

- HCA handle.
- QP handle.

Output Modifiers:

- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid QP handle.
  - QP still has Type 2A MWs bound to it.

- The QP is still attached to one or more multicast groups.

11.2.5 GET SPECIAL QP

Description:

Returns the handle for the specified QP type for the specified HCA port. The special QP types include: SMI QP (QP0), GSI QP (QP1), Raw IPv6 and Raw Ethertype.

C11-13: This Verb **must** support QP0 and QP1.

HCA support for both Raw Datagram types is optional.

o11-1: If the HCA supports the Raw Datagram QP types, this Verb **must** also support them.

C11-14: Handles associated with the SMI QP and the GSI QP **must** only be given out once for each QP per HCA port. Subsequent invocations of this Verb, without an intervening Destroy QP, **must** return an error.

The single QP per port restriction does not apply to either Raw Datagram QP types.

o11-2: If Raw Datagram Service is supported, the number of Raw Datagram type QPs supported per port **shall** be returned by the Query HCA Verb.

Any fixed QP attributes for the specified QP type required by the specific implementation are set up before returning from this Verb. For example, the appropriate Transport Service Type may need to be initialized for the QP.

C11-15: SMI/GSI QPs **shall not** share a completion queue with any non-SMI/GSI QP. An attempt to do so **shall** result in an Invalid CQ Handle error.

Input Modifiers:

- HCA Handle.
- HCA port number.
- The QP type requested. The allowed types are:
  - SMI QP (QP0).
  - GSI QP (QP1).
  - Raw IPv6.
  - Raw Ethertype.
- The CQ to be associated with the Send Queue.
- The CQ to be associated with the Receive Queue.

- The maximum number of outstanding Work Requests the Consumer expects to submit to the Send Queue.
- The maximum number of outstanding Work Requests the Consumer expects to submit to the Receive Queue.
- The maximum number of scatter/gather elements the Consumer expects to specify in a Work Request submitted to the Send Queue.
- The maximum number of scatter/gather elements the Consumer expects to specify in a Work Request submitted to the Receive Queue.
- The Signaling Type for the Send Queue on this QP. The valid types are:
  - All Work Requests submitted to the Send Queue always generate a completion entry.
  - Consumer must specify on each Work Request submitted to the Send Queue whether to generate a completion entry for successful completions.
- Protection Domain.

Output Modifiers:

- QP handle.
- The actual number of outstanding Work Requests supported on the Send Queue. If an error is not returned, this is guaranteed to be greater than or equal to the number requested. (This may require the Consumer to increase the size of the CQ.)
- The actual number of outstanding Work Requests supported through the Verbs on the Receive Queue. If an error is not returned, this is guaranteed to be greater than or equal to the number requested. (This may require the Consumer to increase the size of the CQ.)
- The actual number of scatter/gather elements that can be specified in Work Requests submitted to the Send Queue. If an error is not returned, this is guaranteed to be greater than or equal to the number requested.
- The actual number of scatter/gather elements that can be specified in Work Requests submitted to the Receive Queue. If an error is not returned, this is guaranteed to be greater than or equal to the number requested.
- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA handle.

- Invalid Special QP type.
- QP already in use (applies only to SMI and GSI QPs).
- Number of available Raw Datagram QPs exceeded.
- Invalid Port.
- Invalid CQ handle.
- Maximum number of Work Requests requested exceeds HCA capability.
- Maximum number of scatter/gather elements requested exceeds HCA capability.
- Invalid Protection Domain.
- Raw Datagrams not supported.

11.2.6 COMPLETION QUEUE

11.2.6.1 CREATE COMPLETION QUEUE

Description:

Creates a CQ on the specified HCA.

The Consumer must specify the minimum capacity of the CQ.

The actual capacity of the specified CQ is returned on successful creation. The number returned differs only when the actual capacity is more than the Consumer requested. If the maximum capacity supported by the HCA is less than the capacity requested, an error is returned.

On success, a handle to the newly created Completion Queue is returned.

Input Modifiers:

- HCA handle.
- The minimum capacity of the CQ.
- Completion Event Handler Identifier. If non-zero, associates the CQ with a Completion Event Handler.

Output Modifiers:

- The handle of the newly created CQ.
- The actual capacity of the CQ.
- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA handle.

- CQ capacity requested exceeds HCA capability.
- Invalid Completion Event Handler Identifier.
- HCA doesn't support Base Queue Management Extensions.

11.2.6.2 QUERY COMPLETION QUEUE

Description:

Returns the capacity of the specified CQ.

Input Modifiers:

- HCA handle.
- CQ handle.

Output Modifiers:

- The capacity of the CQ.
- Completion Event Handler Identifier. If zero, no Completion Event Handler is associated with the CQ.
- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid CQ handle.
  - CQ has overrun or has become inaccessible.

11.2.6.3 RESIZE COMPLETION QUEUE

Description:

Modifies the capacity of the CQ.

C11-16: The CI **must** support resizing a CQ with outstanding Work Completions and while Work Requests are outstanding on queues associated with the specified CQ. Completions **must not** be lost as a result of a resize.

The resize operation is allowed to adversely affect the performance while the CQ is being resized. The act of resizing is not allowed to directly generate completion or asynchronous errors.

Input Modifiers:

- HCA handle.
- CQ handle.
- The minimum capacity of the CQ.

Output Modifiers:

- The actual capacity of the CQ.
- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA handle.
  - Invalid CQ handle.
  - CQ capacity requested exceeds HCA capability.
  - More outstanding entries on CQ than capacity specified.
  - CQ has overrun or has become inaccessible.

11.2.6.4 DESTROY COMPLETION QUEUE

Description:

Destroys the specified CQ. Resources allocated by the Channel Interface to implement the CQ must be deallocated during the destroy operation.

C11-17: The CI **shall** return an error if this Verb is invoked while a Work Queue is still associated with the CQ.

Any completions that have not been retrieved from the CQ prior to being destroyed are discarded.

Input Modifiers:

- HCA handle.
- CQ handle.

Output Modifiers:

- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid CQ handle.
  - One or more Work Queues is still associated with the CQ.

11.2.7 EE CONTEXT

11.2.7.1 CREATE EE CONTEXT

Description:

Creates an EE Context for the specified HCA.

On success, a handle to the newly created EE Context is returned.

The values for Remote Node Address Handle, Send Sequence Number, Receive Sequence number are all zero. The EE Context is created in the Reset state.

Input Modifiers:

- HCA handle.
- Reliable Datagram Domain.

Output Modifiers:

- The handle for the newly created EE Context.
- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA handle.
  - Reliable Datagrams not supported.
  - Invalid Reliable Datagram Domain.

11.2.7.2 MODIFY EE CONTEXT ATTRIBUTES

Description:

o11-3: If the CI supports RD Service, upon invocation of this Verb the CI **shall** modify the attributes for the specified EE Context and then **shall** cause the EE Context to transition to the specified EE Context state.

Only a subset of the attributes can be modified once the EE Context has been created.

EE Context attributes can be modified with Work Requests outstanding which use the EE handle, but any such Work Requests might not execute correctly if the modifiers are changed.

WRs can be outstanding on the Receive Queue when the EE is in the Init, RTR, and RTS state and on the Send Queue when the EE is in the RTS state. Any outstanding Work Requests on the QP may not execute as expected if the EE modifiers are changed.

If any of the EE attributes to be modified are invalid or the requested state transition is invalid, none of the EE attributes shall be modified. An immediate error shall be returned and the EE state shall remain unchanged.

o11-4: If the CI supports RD Service, the properties and requirements of the EE Context state transitions **shall** be supported as shown in Table 92.

Table 92 EE Context State Transition Properties

Transition	Required Attributes	Optional Attributes	Actions
Reset to Init	Primary Physical port. P_Key index.	None.	Enable posting to the receive queue.
Init to Init	None.	Primary Physical port. P_Key index.	No transition.
Init to RTR	Remote Node Address Vector. Destination EEC Number. RQ PSN. Number of responder resources for RDMA Read/atomic ops. (Note this is 1 for this revision.)	Alternate path address information. P_Key index.	Activate receive processing.
RTR to RTS	Local ACK Timeout. Retry count. RNR retry count. SQ PSN. Number of Outstanding RDMA Read/atomic ops at destination.	Alternate path address information. Path migration state.	Activate send processing.
RTS to RTS (no transition)	None.	Alternate path address information. Path migration state.	No transition.
RTS to SQD	None.	None.	Deactivate send processing.

Table 92 EE Context State Transition Properties (Continued)

Transition	Required Attributes	Optional Attributes	Actions
SQD to SQD	None.	Remote Node Address Vector (Connected QPs only).[a] [b] Alternate path address information. Path migration state. Local ACK Timeout.[a] Retry count.[a] RNR retry count.[a] Number of Outstanding RDMA Read/atomic ops at destination.[a] Number of local RDMA Read/atomic responder resources.[a] P_Key index.[a] Primary physical port associated with EE if HCA supports this capability.[a]	Modify EE attributes
SQD to RTS	None	Alternate path address information. Path migration state.	Activate send processing.
Any State to Error	None.	None allowed.	Queue processing is stopped. Work Requests in process are completed in error, when possible.
Any state to Reset	None.	None allowed.	EE attributes are reset to the same values after the EE was created. Outstanding Work Requests are removed from the queues without notifying the Consumer.

a. It is allowed to change this attribute only when the SQ is drained.
b. When changing the Remote Node Address Vector, the Path MTU cannot be changed in the SQD2SQD transition.

Input Modifiers:

- HCA handle.
- EE Context handle.

- The EE Context attributes to modify and their new values. The EE Context attributes that can be modified after the EE Context has been created are:
  - Primary physical port. Applicable for the Init to Init transition. Applicable for the SQD to RTS transition, if supported by the HCA.
  - Primary path P_Key Index.
  - PSNs for Sends & Receives.
    - EE Context State.Enable or disable Send Queue Drained, Asynchronous Affiliated Event Notification. This modifier is only applicable when the next EE state chosen is SQD.
  - Initiator Depth: Number of RDMA Reads & Atomic Operations outstanding at any time on the destination EE.
  - Responder Resources: Number of responder resources for handling incoming RDMA Reads & atomic operations. This value may be rounded up to a supported value, not to exceed the maximum value allowable for EEs for this HCA. Note for this version of the specification, this value is one.
  - Destination EE Context number
  - Primary Address vector, containing:
    - Service level.
    - Send Global Routing Header Flag.
    - Destination LID. If destination is in same subnet, LID = final destination; otherwise LID = router LID.
    - Path MTU.
    - Maximum Static Rate.
    - Local ACK Timeout (see Sections 12.7.33 Target ACK Delay on page 678 and 12.7.34 Local ACK Timeout on page 678).
    - Retry count.
    - RNR retry count.
    - Source Path Bits.
    - For global destination:
      - Traffic class.
      - Flow label.
      - Hop limit.
      - Source GID index.
      - Destination's GID.

- Alternate path address information. Valid only when automatic path migration is enabled.
  - Alternate path P_Key index. Note that the alternate path P_Key index should map to the same partition key as the primary P_Key index; otherwise APM may not function properly.
  - Alternate physical port.

  Alternate path address vector, containing:
  - Service level.
  - Send Global Routing Header Flag.
  - Destination LID. If the destination is in the same subnet, LID = final destination; otherwise LID = router LID.
  - Maximum Static Rate.
  - Local ACK Timeout (see Sections 12.7.33 Target ACK Delay on page 678 and 12.7.34 Local ACK Timeout on page 678).
  - Source Path Bits.

  For global destination:
  - Traffic class.
  - Flow label.
  - Hop limit.
  - Source GID index.
  - Destination's GID.
- Path migration state. Valid only if this HCA supports automatic path migration. Valid states to set are:
  - Migrated.
  - Rearm.

Output Modifiers:

- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA handle.
  - Invalid EE Context handle.
  - Cannot change EE Context attribute.
  - Invalid EE Context state.
  - Invalid path migration state.

- Reliable Datagrams not supported.

11.2.7.3 QUERY EE CONTEXT

Description:

Returns the attribute list and current values for the specified EE Context.

Input Modifiers:

- HCA handle.
- EE Context handle.

Output Modifiers:

- The EE Context attributes. The list of attributes returned by the query are:
  - Current EE Context State. Where the state is one of the following:
    - Reset
    - Initialized
    - Ready to Receive (RTR)
    - Ready to Send (RTS)
    - SQ Error
    - SQ Drain (SQD)
      The following modifiers are valid only when the EE Context is in the SQD state:
      - Send Queue Draining
      - Send Queue Drained
    - Error
  - EE Context Number.

  The following attributes are not defined if the EE is in the Reset state.

  - Primary physical port.
  - Primary path P_Key Index.
  - PSNs for Sends & Receives.
  - Reliable Datagram Domain.
  - Initiator Depth: Number of RDMA Reads & Atomic Operations outstanding at any time on the destination EE.
  - Responder Resources: Number of responder resources for handling incoming RDMA Reads & atomic operations.

- Destination EE Context number.
- Primary Address vector, containing:
  - Service level.
  - Send Global Routing Header Flag.
  - Destination LID. If destination is in same subnet, LID = final destination; otherwise LID = router LID.
  - Path MTU.
  - Maximum Static Rate.
  - Local ACK Timeout (see Sections 12.7.33 Target ACK Delay on page 678 and 12.7.34 Local ACK Timeout on page 678).
  - Retry count.
  - RNR retry count.
  - Source Path Bits.
  - For global destination:
    - Traffic class.
    - Flow label.
    - Hop limit.
    - Source GID index.
    - Destination's GID.
- Alternate path address information. Valid only when automatic path migration is enabled.
  - Alternate path P_Key index.
  - Alternate physical port.

  Alternate path address vector, containing:
  - Service level.
  - Send Global Routing Header Flag.
  - Destination LID. If the destination is in the same subnet, LID = final destination; otherwise LID = router LID.
  - Maximum Static Rate.
  - Local ACK Timeout (see Sections 12.7.33 Target ACK Delay on page 678 and 12.7.34 Local ACK Timeout on page 678).
  - Source Path Bits.

  For global destination:
    - Traffic class.
    - Flow label.

- Hop limit.
- Source GID index.
- Destination's GID.

- Path migration state. Applicable only if this HCA supports automatic path migration.

- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid EE Context handle.
  - Reliable Datagrams not supported.

11.2.7.4 DESTROY EE CONTEXT

Description:

Destroys the specified EE Context. Any resources allocated by the Channel Interface for use by the EE Context are freed from use.

o11-5: If the CI supports RD Service, after this Verb is invoked, any outstanding or subsequently submitted Work Requests which depend on the EE Context **shall** complete with an Invalid EE Context Number error.

Input Modifiers:

- HCA handle.
- EE Context handle.

Output Modifiers:

- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid EE Context handle.
  - Reliable Datagrams not supported.

11.2.8 MEMORY MANAGEMENT

Memory Management Verbs are partitioned into two categories:

1) Memory Region Verbs
   - Allocate L_Key.
   - Register Memory Region.
   - Register Physical Memory Region.
   - Query Memory Region.

- Deregister Memory Region.
- Reregister Memory Region.
- Reregister Physical Memory Region.
- Register Shared Memory Region.
- Fast Register Physical MR (a.k.a. Post Send Register Non-Shared MR)
- Local Invalidate
- Send with Invalidate

2) Memory Window Verbs

- Allocate Memory Window.
- Query Memory Window.
- Bind Memory Window.
- Post Send Bind Memory Window
- Deallocate Memory Window.

11.2.8.1 ALLOCATE L_KEY

Description:

Allocates physical buffer list resources for use in memory registrations. Note, the Consumer owns the key portion of the L_Key and R_Key, if any was requested, that is returned from this verb.

This verb either completes entirely or not at all.

Input Modifiers:

- HCA Handle.
- Protection Domain.
- Requested size of the PBL resources to be allocated.
- Remote Access Requested. Requests that an R_Key be returned.

Output Modifiers:

- Memory Region Handle - used to identify this specific registered region to the Memory Management Verbs.
- L_Key - used for local access.
- R_Key - used for remote access.

  The R_Key is returned only when Remote Access was requested.
- Actual size of the PBL resources allocated. The actual size of the PBL resources allocated must be equal to or greater than the size of the PBL resources requested.

- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA handle.
  - Invalid Protection Domain.
  - HCA doesn't support Base Memory Management Extensions.

11.2.8.2 REGISTER MEMORY REGION

Description:

Prepares a virtually addressed memory region for use by an HCA. A description of the registered memory suitable for use in Work Requests to describe locally accessible memory locations is returned. When specifically requested, a description of the registered memory suitable for use by inbound RDMA and/or atomic operations is returned.

This Verb depends on OSV supplied functions to perform the pinning of memory pages and creating the virtual to physical translations that represent the memory region.

The L_Key, and if requested R_Key, returned from this verb are owned by the CI.

Input Modifiers:

- HCA Handle.
- Virtual Address - the address of the first byte of the region to be registered. The Maximum size of a Virtual Address is 64 bits.
- Length of region to be registered in bytes.
- Protection Domain to be assigned to the registered region.
- Access Control - The following may be selected in any combination except as noted.
  - Enable Local Write Access.
  - Enable Remote Write Access.

    Remote Write Access requires Local Write Access to be enabled.
  - Enable Remote Read Access.
  - Enable Remote Atomic Operation Access (If Atomic Ops supported).

    Remote Atomic Operation Access requires Local Write Access.
  - Enable Memory Window Binding.

Note: Local Read Access is always implied.

- Type of VA:
  - Virtual Address
  - Zero Based Virtual Address.

Output Modifiers:

- Memory Region Handle - used to identify this specific registered region to the Memory Management Verbs.
- L_Key - used for local access.
- R_Key - used for remote access.

  The R_Key is returned only when Remote Access was requested.
- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA handle.
  - Invalid Virtual Address.
  - Invalid Length
  - Invalid Protection Domain.
  - Invalid Access Control specifier.
  - HCA doesn't support ZBVA.

11.2.8.3 REGISTER PHYSICAL MEMORY REGION

Description:

Prepares a physically addressed Memory Region for use by an HCA. A description of the registered memory suitable for use in Work Requests to describe locally accessible memory locations is returned. When specifically requested, a description of the registered memory suitable for use by inbound RDMA and/or atomic operations is returned.

In addition to a list of physical buffers, the Consumer supplies a requested "I/O Virtual Address" to be associated with the first byte of the Region. The Consumer also supplies the length of the entire Region plus a byte offset that specifies where the Region begins within the first physical buffer.

If the CI doesn't support the Base Memory Management extensions, the Channel Interface returns the I/O Virtual Address that is actually assigned for the Region. If the CI supports the Base Memory Management extensions, the CI returns the I/O Virtual Address supplied by the Consumer.

Input Modifiers:

- HCA Handle.
- Key ownership requested. Consumer requests ownership of the key portion of L_Key and R_Key, if any was requested. Note, if the Consumer doesn't request ownership of the key portion, then the L_Key and R_Key, if any was requested, returned from this verb cannot be used in a Fast Register Physical MR.
  - Key to use on the new L_Key and R_Key.
- Physical Buffer List.

  Starting physical address of each physical buffer.
  - If the PBL is a Page Type, each buffer must begin and end on an HCA-supported page boundary.
  - If the PBL is a Block Type, each buffer may begin at an arbitrary physical address

  Page size. Used only if the HCA supports multiple page sizes per MR.
- Physical buffer size. All Block Sizes in the list must be the same for Block Type Physical Buffers. All Page Sizes in the list must be the same, if the HCA doesn't support multiple page sizes per MR. Not applicable if the HCA supports multiple page sizes per MR. A buffer size must match one of the buffer sizes supported by the HCA.
- Total number of Physical Buffers in the list.
- Type of VA:
  - Virtual Address
  - Zero Based Virtual Address.
- If VA type is VA, the IOVA requested by the Consumer for the first byte of the region. Note, for ZBVA no IOVA is passed.
- Length of Region to be registered in bytes.
- Offset of Region's starting IOVA within the first physical buffer.
- Protection Domain to be assigned to the registered region.
- Access Control - The following may be selected in any combination except as noted.
  - Enable Local Write Access.
  - Enable Remote Write Access.

    Remote Write Access requires Local Write Access to be enabled.
  - Enable Remote Read Access.

- Enable Remote Atomic Operation Access (If Atomic Ops supported).

  Remote Atomic Operation Access requires Local Write Access.
- Enable Memory Window Binding.

  Note: Local Read Access is always implied.

Output Modifiers:

- Memory Region Handle - used to identify this specific registered region to the Memory Management Verbs.
- I/O Virtual Address - IOVA actually assigned by the Channel Interface for the first byte of the Region. The IOVA is returned if this verb completes successfully.
- L_Key - used for local access.
- R_Key - used for remote access. The R_Key is returned if the Access Control input modifier has any of the Remote Accesses enabled.
- Actual size of the PBL resources allocated. The actual size of the PBL resources allocated shall be greater than or equal to the size of the PBL resources passed in by the Consumer as an input modifier.
- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA handle.
  - Invalid Physical Buffer List entry.
  - Invalid Length.
  - Invalid Offset.
  - Invalid Protection Domain.
  - Invalid Access Control specifier.
  - HCA doesn't support Base Memory Management Extensions.
  - HCA doesn't support ZBVA.
  - HCA was not opened in Page mode.
  - HCA was not opened in Block mode.
  - HCA doesn't support multiple PB sizes per MR.

11.2.8.4 QUERY MEMORY REGION

Description:

Retrieves information about a specific memory region.

Input Modifiers:

- HCA Handle.
- Memory Region Handle - as issued when region was registered.

Output Modifiers:

- L_Key.
- R_Key. R_Key is returned only if it was previously allocated.
- L_Key state.
- Ownership attributes for the key portion of L_Key and R_Key, if any R_Key exists for the Memory Region: Consumer owns vs. CI owns.
- Sharing attributes of the Memory Region: Shared vs. Not Shared.
- Actual number of allocated Physical Buffer List entries for Physical Memory Region referenced by the L_Key or R_Key.
- Type of VA:
  - Virtual Address
  - Zero Based Virtual Address.
- Actual Local Protection Bounds enforced by the Channel Interface.
- Actual Remote Protection Bounds enforced by the Channel Interface.

  The Remote Protection Bounds are returned only when Local and Remote Access to the region was requested.
- Protection Domain assigned to the registered region.
- Access Control settings for the registered region.
- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle
  - Invalid Memory Region handle.

11.2.8.5 DEREGISTER MEMORY REGION

Description:

Removes a memory region from the HCA translation table. The region is unpinned if pinned in the associated registration Verb. This Verb is responsible only for deallocating resources allocated as part of the associated registration operation. All other resources are the responsibility of the Consumer.

It is an error for a Consumer to attempt to deregister a Memory Region while it still has any Memory Windows bound to it. Channel Interface implementations have options on how to deal with the error, described in 10.6.7.2.6 Deregistering Regions with Bound Windows on page 502.

Work Requests or Remote Operation requests that are in process and actively referencing memory locations in a Memory Region that is deregistered must fail with a protection violation. Work Requests or Remote Operation requests that attempt to access memory locations in a Memory Region that has been deregistered must fail with a protection violation.

This Verb depends on the availability of OSV supplied functions to perform the unpinning of memory pages.

Input Modifiers:

- HCA Handle.
- Memory Region Handle - as issued when region was registered.

Output Modifiers:

No output modifiers.

- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid Memory Region handle.
  - Operation denied; Region still has bound Window(s)

11.2.8.6 REREGISTER MEMORY REGION

Description:

Modifies the attributes of an existing Memory Region. Any existing Region owned by the Consumer can be modified, regardless of which Verb created it initially[16], or which Verb (if any) reregistered it most recently[17]. A description of the Memory Region suitable for use in Work Requests to describe locally accessible memory locations is returned. When specifically requested, a description of the Memory Region suitable for use by inbound RDMA and/or atomic operations is returned.

This Verb conceptually performs the functions Deregister Memory Region followed by Register Memory Region. Where possible, resources below the Verb layer are expected to be reused instead of deallocated

16. For instance, a Region created with Register *Physical* Memory Region can later be modified by Reregister Memory Region.

17. For instance, a Region modified by Reregister *Physical* Memory Region can later be modified by Reregister Memory Region.

and reallocated. This Verb may be used to change the access rights and/or protection domain of a region, as well as changing the memory locations that are registered.

The L_Key and R_Key output modifiers from this Verb must be used in place of any previously issued for this region.

This Verb depends on the availability of OSV supplied functions to perform the pinning and unpinning of memory pages and creating the virtual to physical translations that represent the memory region.

It is an error for a Consumer to attempt to reregister a Memory Region while the Region still has any Memory Windows bound to it. Channel Interface implementations have options on how to deal with the error, as described in 10.6.7.2.6 Deregistering Regions with Bound Windows on page 502.

C11-18: If the CI returns the "Operation denied" (due to a bound Window) error, the CI **shall** make no change to the current registration.

C11-19: If the CI returns either the "Invalid HCA handle" or "Invalid Memory Region handle" error, the CI **shall** make no change to the current registration (assuming that it even exists).

C11-20: If the CI returns any other error, the CI **shall** invalidate both "old" and "new" registrations, and release any associated resources.

C11-21: For the error case where a remote agent is accessing a Memory Region while it is in the process of being reregistered, the CI **must** present the same semantics as a deregistration operation followed by a separate registration operation.

The L_Key, and if requested R_Key, returned from this verb are owned by the CI. Note, the L_Key, and any accompanying R_Key, that is passed in as an input modifier may have been owned by the Consumer. However, upon successful completion of this verb the CI owns the key portion of the L_Key and R_Key, if any was requested.

A shared MR becomes a non-shared MR upon successful completion of this verb, if the Change Translation Input Modifier is set. Otherwise the shared MR remains a shared MR.

Input Modifiers:

- HCA Handle.
- Memory Region Handle - as issued when region was registered.
- Change Request type - The following may be selected in any combination, the input modifiers required to support the request are listed below each request.
  - Change Translation.

Input Modifiers required.

- Virtual Address.
- Length.
- Type of VA:
  - Virtual Address
  - Zero Based Virtual Address.

- Change Protection Domain.

  Input Modifiers required.

  - Protection Domain.
- Change Access Control.

  Input Modifiers required.

  - Access Control Selections.

Output Modifiers:

- Memory Region Handle - must be used for future references to this Memory Region. Might or might not be the same as the previous Region Handle.
- L_Key - used for local access.
- R_Key - used for remote access.

  The R_Key is returned only when Remote Access was requested.
- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA handle.
  - Invalid Memory Region handle.
  - Invalid Virtual Address.
  - Invalid Length.
  - Invalid Protection Domain.
  - Invalid Access Control specifier.
  - Operation denied; Region still has bound Window(s).
  - HCA doesn't support ZBVA.

Usage Example:

a) To modify only the Access Control of an already registered region, the Memory Region Handle, a Change Access Control Request and the new Access Control Selections input modifiers would be supplied to the Verb.

b) To change the address translations of a region the Memory Region Handle, a Change Translation Request and the new Virtual Address and length input modifiers would be supplied to the Verb. The pages previously pinned would be unpinned, the new memory region would be pinned and registered (and if requested bound) using the region's access controls and protection domain. Previous translations would be removed or replaced as needed.

11.2.8.7 REREGISTER PHYSICAL MEMORY REGION

Description:

Modifies the attributes of an existing Memory Region. Any existing Region owned by the Consumer can be modified, regardless of which Verb created it initially[18], or which Verb (if any) reregistered it most recently[19]. A description of the Memory Region suitable for use in Work Requests to describe locally accessible memory locations is returned. When specifically requested, a description of the Memory Region suitable for use by inbound RDMA and/or atomic operations is returned.

This Verb conceptually performs the functions Deregister Memory Region followed by Register Physical Memory Region. Where possible, resources below the Verb layer are expected to be reused instead of deallocated and reallocated. This Verb may be used to change the access rights and/or protection domain of a region, as well as changing the memory locations that are registered.

Note, the L_Key, and any accompanying R_Key, that is passed in as an input modifier may have been owned by CI. If the Consumer requests ownership of the key portion of L_Key and R_Key, if any was requested, then upon successful completion of this verb the Consumer would own the key portion of the L_Key and R_Key, if any was requested.

The L_Key and R_Key output modifiers from this Verb must be used in place of any previously issued for this region. The CI can use any of the following three input modifiers to access the Memory Region: Memory Region Handle, L_Key, or R_Key. The CI is not expected to perform consistency checks between these three input modifiers.

It is an error for a Consumer to attempt to reregister a Memory Region while the Region still has any Memory Windows bound to it. Channel Interface implementations have options on how to deal with the error, as described in 10.6.7.2.6 Deregistering Regions with Bound Windows on page 502.

18. For instance, a Region created with Register Memory Region can later be modified by Reregister *Physical* Memory Region.

19. For instance, a Region modified by Reregister Memory Region can later be modified by Reregister *Physical* Memory Region.

C11-22: For the Reregister Physical Memory Region Verb, the CI shall conform to all of the compliance statements contained in 11.2.8.6 Reregister Memory Region on page 599.

A shared MR becomes a non-shared MR upon successful completion of this verb, if the Change Translation Input Modifier is set. Otherwise the shared MR remains a shared MR.

Input Modifiers:

- HCA Handle.
- Memory Region Handle
- L_Key
- R_Key. R_Key is supplied only if the L_Key has an accompanying R_Key.
- Change Request type - The following may be selected in any combination, the input modifiers required to support the request are listed below each request.
  - Change Translation. Input modifiers required:
    - Key ownership requested. Consumer requests ownership of the key portion of L_Key and R_Key, if any was requested. Note, if the Consumer doesn't request ownership of the key portion, then the L_Key and R_Key, if any was requested, returned from this verb cannot be used in a Fast Register Physical MR.
      - Key to use on the new L_Key and R_Key.
    - Physical Buffer List.

      Starting physical address of each physical buffer.
      - If the PBL is a Page Type, each buffer must begin and end on an HCA-supported page boundary.
      - If the PBL is a Block Type, each buffer may begin at an arbitrary physical address

      Page size. Used only if the HCA supports multiple page sizes per MR.
    - Physical buffer size. All Block Sizes in the list must be the same for Block Type Physical Buffers. All Page Sizes in the list must be the same, if the HCA doesn't support multiple page sizes per MR. Not applicable if the HCA supports multiple page sizes per MR. A buffer size must match one of the buffer sizes supported by the HCA.
    - Total number of Physical Buffers in the list.
    - Type of VA:

- Virtual Address
- Zero Based Virtual Address.

- If VA type is VA, the IOVA requested by the Consumer for the first byte of the region. Note, for ZBVA no IOVA is passed.
- Length of Region to be registered in bytes.
- Offset of Region's starting IOVA within the first physical buffer.

- Change Protection Domain.
  - Protection Domain to be assigned to the registered region
- Change Access Control

  Access Control - The following may be selected in any combination except as noted.
  - Enable Local Write Access.
  - Enable Remote Write Access.

    Remote Write Access requires Local Write Access to be enabled.
  - Enable Remote Read Access.
  - Enable Remote Atomic Operation Access (If Atomic Ops supported).

    Remote Atomic Operation Access requires Local Write Access.
  - Enable Memory Window Binding.

    Note: Local Read Access is always implied.

Output Modifiers:

- Memory Region Handle - used to identify this specific registered region to the Memory Management Verbs.
- I/O Virtual Address - IOVA actually assigned by the Channel Interface for the first byte of the Region. For an HCA that supports the Base Memory Management Extensions, the returned IOVA is the same as the IOVA requested.
- L_Key - used for local access.
- R_Key - used for remote access. The R_Key is returned if the Access Control input modifier has any of the Remote Accesses enabled.
- Actual size of the PBL resources allocated. The actual size of the PBL resources allocated shall be greater than or equal to the size of the PBL resources passed in by the Consumer as an input modifier.

- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA handle.
  - Invalid Physical Buffer List entry.
  - Invalid Length.
  - Invalid Offset.
  - Invalid Protection Domain.
  - Invalid Access Control specifier.
  - HCA doesn't support Base Memory Management Extensions.
  - HCA doesn't support ZBVA.
  - HCA was not opened in Page mode.
  - HCA was not opened in Block mode.
  - HCA doesn't support multiple PB sizes per MR.

11.2.8.8 REGISTER SHARED MEMORY REGION

Description:

Given an existing Memory Region, a new independent Memory Region associated with the same physical memory locations is created, with the intention that the new Memory Region share HCA mapping resources to the extent possible. Through repeated calls to the Verb, an arbitrary number of Memory Regions can potentially share the same HCA mapping resources, all associated with the same physical memory locations. The memory region created by this verb behaves identically to memory regions created by the other memory registration verbs.

The Virtual Address, Protection Domain, and Access Rights specified for the new Memory Region need not be the same as those of the existing Memory Region. The lengths are by definition the same.

The Consumer supplies a requested Virtual Address to be associated with the first page in the new Memory Region, and the Channel Interface returns the Virtual Address that is actually assigned.

The L_Key, and if requested R_Key, returned from this verb are owned by the CI.

Input Modifiers:

- HCA Handle.
- Memory Region Handle - of an already registered region.

- Virtual Address - requested by the Consumer for the first page of the buffer.
- Protection Domain.
- Access Control Selections.
- Type of VA:
  - Virtual Address
  - Zero Based Virtual Address.

Output Modifiers:

- Memory Region Handle - of the new Memory Region.
- Virtual Address - actually assigned by the Channel Interface for the first page.
- L_Key - used for local access.
- R_Key - used for remote access.

  The R_Key is returned when Remote Access Rights are requested.
- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA handle.
  - Invalid Memory Region handle.
  - Invalid Protection Domain.
  - Invalid Access Control specifier.
  - HCA doesn't support ZBVA.

11.2.8.9 ALLOCATE MEMORY WINDOW

Description:

This Verb allocates a memory window which is associated with a protection domain. It is not inherently associated with any memory region when allocated.

Input Modifiers:

- HCA Handle.
- Protection Domain to be assigned to the Memory Window.
- Type of Memory Window: Type 1 or Type 2.

Output Modifiers:

- Window Handle - used to identify this specific Memory Window to other Memory Management Verbs.
- R_Key - an unbound R_Key for use in specifying the Window with the Bind Memory Window Verb.
- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA handle.
  - Invalid Protection Domain.
  - HCA doesn't support Type 2 Memory Windows.

11.2.8.10 QUERY MEMORY WINDOW

Description:

This Verb returns the attributes associated with the specified memory window.

Input Modifiers:

- HCA Handle.
- Window Handle - as issued by an Allocate Memory Window.

Output Modifiers:

- R_Key - the current R_Key associated with the Memory Window.
- Type of Memory Window: Type 1 or Type 2.
- R_Key state.
- Protection Domain associated with the Memory Window.
- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid Memory Window handle.

11.2.8.11 BIND MEMORY WINDOW

Description:

Posts a Work Request to a specified Send Queue, which binds a Memory Window to a specified VA range and remote access attributes based on an existing Memory Region. The QP Service Type must be either Reliable Connection, Unreliable Connection, or Reliable Datagram.

The specified VA range must either be the entire Memory Region or a subset of it. Remote Write Access or Remote Atomic Access must not

be specified unless the Memory Region has Local Write Access. The QP, Memory Window, and Memory Region must belong to the same HCA and Protection Domain.

A previously bound Memory Window can be bound to a new VA range in the same or a different Memory Region, causing the previous binding to be invalidated. Binding a previously bound Memory Window to a zero-length VA range will invalidate the previous binding and return an R_Key that is in the unbound state.

The Bind operation has a unique ordering rule: any Work Request posted to a Send Queue subsequent to a Bind must not begin execution until the Bind operation completes.

Under normal operation, it is improper for a Consumer to change the binding of a Memory Window while it is being accessed by a remote agent. However, this can occur if remote agents misbehave, or it can occur under error recovery circumstances. Any Remote Operation requests that are in process and actively using a Memory Window when its binding is changed must fail with a protection violation. Once the Bind operation has been reported to the Consumer as having completed, the Channel Interface must guarantee that no additional accesses can be performed under the immediate previous binding.

If the bind memory window could not be completed due to an invalid input modifier, the previous bind memory window settings shall remain valid and none of the previous window's attributes shall be modified. An immediate or completion error shall be returned.

Input Modifiers:

- HCA Handle.
- QP Handle.
- The Work Request containing the information required to perform the request. The Work Request is defined as follows:
  - A user defined 64-bit Work Request ID.
  - Memory Window Handle.
  - R_Key - The R_Key currently associated with the Memory Window.
  - Memory Region Handle.
  - L_Key - The L_Key for the Memory Region that the Memory Window will be associated with.
  - Virtual Address - the address of the first byte of the bound range. The Maximum size of a Virtual Address is 64 bits.
  - Length of range to be bound in bytes.
  - Access Control - The following may be selected in any combination except as noted.

- Enable Remote Write Access. Requires the Memory Region to have Local Write Access.
- Enable Remote Read Access
- Enable Remote Atomic Operation Access (If Atomic Ops supported). Requires the Memory Region to have Local Write Access.

- Completion Notification Indicator. Must be specified if the Send Queue was created with a Signaling type of Selectable. Ignored if the Send Queue was created with a Signaling type of Non-Selectable.
- Fence indicator. If the fence indicator is set, then all prior RDMA Read and Atomic Work Requests on the queue must be completed before starting to process this Work Request. The Fence indicator only has an effect with the Reliable Connection and Reliable Datagram transport services.

Output Modifiers:

- R_Key - The R_Key associated with the new binding, whose value is different from that of the supplied R_Key.
- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA handle
  - Invalid QP handle.
  - Invalid Service Type for this QP.
  - Invalid Memory Window handle.
  - Invalid R_Key.
  - Invalid Memory Region handle.
  - Invalid L_Key.
  - Invalid Virtual Address
  - Invalid Length.
  - Invalid Access Control specifier.
  - Too many Work Requests posted.
  - Invalid MW Type. Consumer attempted to perform a Bind operation on a Type 2 Memory Window.

11.2.8.12 DEALLOCATE MEMORY WINDOW

Description:

Under normal operation, it is improper for a Consumer to deallocate a Memory Window while it is being accessed by a remote agent. However, this can occur if remote agents misbehave, or it can occur under error recovery circumstances. Any Remote Operation requests that are in process and actively using a Memory Window when it is deallocated must fail with a protection violation. Once the deallocation Verb completes, the Channel Interface must guarantee that no additional accesses can be performed through that Memory Window.

Input Modifiers:

- HCA Handle.
- Window Handle - as issued by an Allocate Memory Window.

Output Modifiers:

No output modifiers

- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle
  - Invalid Memory Window handle.

11.3 MULTICAST

11.3.1 ATTACH QP TO MULTICAST GROUP

Description:

Attaches the QP to the specified multicast group. The only function of this Verb is to assign the Receive Work Queue of this QP to the specified multicast group; after the attachment completes, this QP will be provided with a copy of every multicast message addressed to the group specified by the MGID and received on the HCA port with which the QP is associated. Creation of the multicast group, and reconfiguration of the fabric such that packets addressed to that group are routed to a local HCA port, is described in 7.10 IBA and Raw Packet Multicast on page 213.

The Service Type of the specified QP must be Unreliable Datagram. It is an error to specify a QP with any other Service Type.

One or more QPs are allowed to be attached to a multicast group on the HCA. If the maximum number of multicast group attachments has already been reached for the HCA when a QP attempts to attach to the multicast group, an error is returned.

If an attempt is made to attach a particular QP to the same multicast group that it is already attached to, the operation will apparently succeed (i.e. return operation completed successfully). However, only

one copy of each multicast message will be delivered to the attached QP.

The input modifier which determines the multicast group to attach to are the MGID and MLID. Both input modifiers must be supplied.

The IBA unreliable multicast feature is optional. This Verb is required only if IBA unreliable multicast is supported by the HCA.

Input Modifiers:

- HCA Handle.
- Multicast group MLID.
- Multicast group MGID.
- QP Handle.

Output Modifiers:

- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA handle.
  - Invalid multicast MLID.
  - Invalid Multicast group MGID.
  - Invalid QP handle.
  - Invalid Service Type for this QP.
  - Number of QPs attached to multicast groups exceeded.

11.3.2 DETACH QP FROM MULTICAST GROUP

Description:

Detaches the specified QP from a multicast group. The only function of this Verb is to detach the Receive Work Queue of this QP from the specified multicast group.

All of the input modifiers must be correct for the QP to be detached. If the QP is attached to a different multicast group or port, an error will be returned.

This Verb is required only if IBA unreliable multicast is supported by the HCA. The IBA unreliable multicast feature is optional.

Input Modifiers:

- HCA Handle.
- Multicast group MLID.
- Multicast group MGID.

- QP Handle.

Output Modifiers:

- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid multicast MLID.
  - Invalid Multicast group MGID.
  - Invalid QP handle.

11.4 WORK REQUEST PROCESSING

11.4.1 QUEUE PAIR OPERATIONS

11.4.1.1 POST SEND REQUEST

Description:

Builds one or more WQEs for the Send Queue in the specified QP from the information contained in the list of Work Requests submitted by the Consumer. These WQEs are added to the end of the Send Queue and the HCA is notified that one or more WQEs are ready to be processed. If the HCA does not support the Base Queue Management Extensions, the CI must support a list of size one.

Note: the Consumer can post up to the remaining capacity of a WQ without encountering a Too Many WRs Posted Immediate Error. If the Consumer posts more than the remaining capacity, an Immediate Error may be returned.

If the Send Queue is enabled for selectable completion notification, for each WR, the Consumer must specify whether a successful completion of the Work Request results in a completion entry on the CQ.

Control returns to the Consumer immediately after the WQEs have been submitted to the Send Queue and the HCA has been notified that one or more WQEs are ready to process. When control returns, the Work Request is in the scope of the Consumer and will no longer be modified or accessed below the Channel Interface. However, for a Fast Register PMR operation the PBL is in the scope of the CI until the Work Request is returned as a Work Completion.

C11-23: If the CI does not support the Base Queue Management Extensions, the CI **shall** return control to the Consumer immediately after the Work Request has been submitted to the Send Queue.

o11-5.2.1: If the CI supports the Base Queue Management Extensions, the CI **shall** return control to the Consumer immediately after the list of Work Request has been submitted to the Send Queue.

C11-24: Once control has been returned to the Consumer the CI **shall not** modify or access any of the Work Requests.

Sends, RDMA and atomic operations can all take place on the same QP. Table 93 Operation Type Matrix shows which operations are allowed for each Service Type of the QP.

o11-5.2.2: For a CI that supports the Base Memory Management Extensions, if a QP is not enabled for Fast Register PMR and Reserved L_Key, and the Consumer attempts a Fast Register PMR or to use the Reserved L_Key, the CI **must** return the appropriate error.

C11-25: This compliance statement has been obsoleted.

C11-25.2.1: The CI **shall** support the operations based on QP Service type according to Table 93 Operation Type Matrix .

Table 93 Operation Type Matrix

	Send	RDMA Read	RDMA Write	Atomic Ops	Type 2 MW Bind[a]	Fast Register Physical MR[a]	Local Invalidate[a]
RC	Yes	Yes	Yes	Yes	Yes	Yes	Yes
RD	Yes	Yes	Yes	Yes	Yes	Yes	Yes
UC	Yes	Not allowed	Yes	Not allowed	Yes	Yes	Yes
UD	Yes	Not allowed	Not allowed	Not allowed	Not allowed	Not allowed	Not allowed
Raw	Yes	Not allowed	Not allowed	Not allowed	Not allowed	Not allowed	Not allowed

a. Only supported if HCA supports Base Memory Management Extensions.

C11-25.2.2: The ordering and fencing considerations for Atomic Operations are the same as for RDMA Read.

Not all of the Input Modifiers are valid for all operations. Table 94 Work Request Modifier Matrix shows which of the Input Modifiers are valid for each operation. If Input Modifiers are specified that are not valid for a particular operation, they are ignored.

C11-26: This compliance statement has been obsoleted.

C11-26.2.1: The CI **shall** ignore all input modifiers in a Work Request that are not valid for the specified operation as shown in Table 94 Work Request Modifier Matrix .

Table 94 Work Request Modifier Matrix[a]

	Send	RDMA Read	RDMA Write	Atomic Ops	Type 2 MW Bind	Fast Register Physical MR	Local Invalidate
Work Request ID	Required	Required	Required	Required	Required	Required	Required
Completion notification indicator	Required if Send Queue Signaling Type is Selectable	Required if Send Queue Signaling Type is Selectable	Required if Send Queue Signaling Type is Selectable	Required if Send Queue Signaling Type is Selectable	Required if Send Queue Signaling Type is Selectable	Required if Send Queue Signaling Type is Selectable	Required if Send Queue Signaling Type is Selectable
Scatter/Gather list	Required[b]	Required[b]	Required[b]	N/A	N/A	N/A	N/A
# of Data Segments	Required[b]	Required[b]	Required[b]	N/A	N/A	N/A	N/A
Immediate Data	Optional except N/A for Raw Datagram QPs	N/A	Optional	N/A	N/A	N/A	N/A
Fence Indicator	Optional for Reliable QPs	Optional for Reliable QPs	Optional for Reliable QPs	Optional for Reliable QPs	Optional for Reliable QPs	N/A	N/A
Remote Node Address	Address Handle Required for UD QPs, DLID, Source Path Bits & SL Required for Raw	N/A	N/A	N/A	N/A	N/A	N/A
Remote Node QP # and Q_Key	Required for IB Datagram QPs[c]	Required for Reliable Datagram QPs	Required for Reliable Datagram QPs	Required for Reliable Datagram QPs	N/A	N/A	N/A
EE Context	Required for Reliable Datagram QPs	Required for Reliable Datagram QP	Required for Reliable Datagram QP	Required for Reliable Datagram QP	N/A	N/A	N/A

Table 94 Work Request Modifier Matrix[a] (Continued)

	Send	RDMA Read	RDMA Write	Atomic Ops	Type 2 MW Bind	Fast Register Physical MR	Local Invalidate
Remote address	N/A	Required	Required	Required	N/A	N/A	N/A
Remote R_Key	Optional for Send w/ Inv. Only for RC.	Required	Required	Required	N/A	N/A	N/A
Atomic operands	N/A	N/A	N/A	Required	N/A	N/A	N/A
Solicited Event	Optional	N/A	Optional with Immediate Data	N/A	N/A	N/A	N/A
Ethertype	Required for Raw Ethertype QPs	N/A	N/A	N/A	N/A	N/A	N/A
Maximum Static Rate	Required for Raw and N/A for others	N/A	N/A	N/A	N/A	N/A	N/A
Local Invalidate Fence	N/A	N/A	N/A	N/A	N/A	N/A	Optional for RC, RD, and UC QPs
New key to use on L_Key and R_Key	N/A	N/A	N/A	N/A	Required	Required	N/A
MR Handle	N/A	N/A	N/A	N/A	Required	Required	Required for MR
MW Handle	N/A	N/A	N/A	N/A	Required	N/A	Required for MW
PBL	N/A	N/A	N/A	N/A	N/A	Required	N/A
First Byte Offset	N/A	N/A	N/A	N/A	N/A	Required	N/A
Addressing Type	N/A	N/A	N/A	N/A	Required	Required	N/A
VA	N/A	N/A	N/A	N/A	Required	N/A	N/A
IOVA	N/A	N/A	N/A	N/A	N/A	Required	N/A
L_Key	N/A	N/A	N/A	N/A	Required	Required	Required for MR

Table 94 Work Request Modifier Matrix[a] (Continued)

	Send	RDMA Read	RDMA Write	Atomic Ops	Type 2 MW Bind	Fast Register Physical MR	Local Invalidate
Local R_Key	N/A	N/A	N/A	N/A	Required	Required	Required
Access Control	N/A	N/A	N/A	N/A	Required	Required	N/A
Length	N/A	N/A	N/A	N/A	Required	Required	N/A

a. Note: If the Service Type is not mentioned in a field in the above table, the modifier is not applicable for that Service Type.
b. Scatter/Gather list is allowed to have zero elements.
c. UD multicast uses a QPN of 0xFFFFFF. See o7-13.1.1:

If an immediate error is returned, the QP state shall not be affected.

Input Modifiers:

This is the full list of modifiers for all of the operations available on the Send Queue. Not all modifiers can be used for all queue or operation types. See Table 93 Operation Type Matrix and Table 94 Work Request Modifier Matrix for details on which modifiers may be used for the specified queue and operation types.

- HCA handle.
- QP handle.
- A list of Work Requests. Where each entry consists a Work Request containing the information required to perform the Work Request. The WR modifiers that must be specified are dependent on the operation type specified. Each Work Request is defined as follows:
  - A user defined 64-bit Work Request ID.
  - Operation type. Valid operation types for Work Requests submitted to the Send Queue are:
    - Send
    - RDMA Read
    - RDMA Write
    - Compare & Swap (assuming the HCA supports atomic operations)
    - Fetch & Add (assuming the HCA supports atomic operations)
    - Fast Register Physical MR (assuming the HCA supports Fast Registration)

  - Local Invalidate (assuming the HCA supports invalidate operations)
  - Type 2 Memory Window Bind (assuming the HCA supports Type 2 Memory Windows)
- (Remote) Invalidate indicator. This indicator will select whether Invalidation will be included in the outgoing Send.
  - R_Key that is to be included in the Send with Invalidate.
- Completion notification indicator. Must be specified and is only valid if the Send Queue was created with a Signaling Type of Selectable.
- Scatter/Gather list. The scatter/gather list can contain zero or more Data Segments. The list is specified only for Send and RDMA operations. Note that for Raw IPv6 QPs, the first 40 bytes of the buffer(s) referred to by the Scatter/Gather list must contain the IPv6 header of the outgoing message.
- Number of Data Segments in the scatter/gather list. This modifier is used only when the scatter/gather list must be specified.
- Immediate Data Indicator. This is set if Immediate Data is to be included in the outgoing request. Valid only for Send or Write RDMA operations.
- 4-byte Immediate Data. Valid only for Send or Write RDMA operations.
- Fence indicator. If the fence indicator is set, then all prior RDMA Read and Atomic Work Requests on the queue must be completed before starting to process this Work Request. The Fence indicator only has an effect with the Reliable Connection and Reliable Datagram transport services.
- Remote node address, required only for operations on Raw or IB Unreliable Datagram Service Types.
- QP number of the destination QP. Required only for operations on IB Datagram Service Types. Use 0xFFFFFF for UD Multicast. See o7-13.1.1:
- The Q_Key for the destination QP. Required only for operations on IB Datagram Service Types. See 10.2.5 Q_Keys on page 439 for more detail on how the CI determines which Q_Key to insert in the packet.
- Ethertype associated with the Work Request. Required only for Raw Ethertype QPs.
- Maximum Static Rate. Required only for Raw IPv6 and Raw Ethertype QPs.

- EE Context. Required only for Reliable Datagram QPs. Note that this is the EE Context number and not the EE Context Handle.
- Solicited Event Indicator. Valid only for RDMA Writes with immediate data or Sends.
- Remote address specified by an address and R_Key. Required and used only for RDMA and atomic operations. For Atomic operations, the address must point to a location that is 64-bit aligned.
- Compare & Swap (atomic) operation operands. If a Compare & Swap operation is specified, the following additional operands must be supplied:
  - Compare operand. Must be 64-bit.
  - Swap operand. Must be 64-bit.
  - A local Data Segment where a copy of the original contents of the remote memory operation will be deposited after the Compare & Swap operation completed at the remote endnode.
- Fetch & Add (atomic) operation operands. If a Fetch & Add operation is specified, the following additional operands must be supplied:
  - Add operand. Must be 64-bit.
  - A local Data Segment where a copy of the original contents of the remote memory operation will be deposited after the Fetch & Add operation completed at the remote endnode.
- Following are the input modifiers specific to the Bind Type 2 Memory Window Work Request:
  - New R_Key Key.
  - Memory Window Handle.
  - R_Key - The R_Key currently associated with the Memory Window.
  - Memory Region Handle.
  - L_Key - The L_Key for the Memory Region that the Memory Window will be associated with.
  - Type of VA:
    - Virtual Address
    - Zero Based Virtual Address.

- Virtual Address - For type 2 MWs, the address of the first byte of the bound range using the same VA base as the underlying MR. The Maximum size of a Virtual Address is 64 bits.
- Length of range to be bound in bytes.
- Access Control - The following may be selected in any combination except as noted.
  - Enable Remote Write Access. Requires the Memory Region to have Local Write Access.
  - Enable Remote Read Access
  - Enable Remote Atomic Operation Access (If Atomic Ops supported). Requires the Memory Region to have Local Write Access.

- Following are the input modifiers specific to the Fast Register Physical MR Work Request:
  - Key to use on the new L_Key and R_Key.
  - L_Key
  - R_Key - should be supplied if R_Key was assigned when MR was allocated.
  - Memory Region Handle
  - Physical Buffer List. It is recommended that the Consumer keeps the PBL available to the CI until the Fast Register Physical MR completes, because the CI may access the PBL until the Work Request is returned as a Work Completion.

    Starting physical address of each physical buffer.
    - If the PBL is a Page Type, each buffer must begin and end on an HCA-supported page boundary.
    - If the PBL is a Block Type, each buffer may begin at an arbitrary physical address

    Page size. Used only if the HCA supports multiple page sizes per MR.
  - Physical buffer size. All Block Sizes in the list must be the same for Block Type Physical Buffers. All Page Sizes in the list must be the same, if the HCA doesn't support multiple page sizes per MR. Not applicable if the HCA supports multiple page sizes per MR. A buffer size must match one of the buffer sizes supported by the HCA.
  - Total number of Physical Buffers in the list.
  - Type of VA:

- Virtual Address
- Zero Based Virtual Address.

- If VA type is VA, the IOVA requested by the Consumer for the first byte of the region. Note, for ZBVA no IOVA is passed.
- Length of Region to be registered in bytes.
- Offset of Region's starting IOVA within the first physical buffer.
- Access Control - The following may be selected in any combination except as noted.
  - Enable Local Write Access.
  - Enable Remote Write Access.

    Remote Write Access requires Local Write Access to be enabled and the L_Key to have an accompanying R_Key.
  - Enable Remote Read Access.

    Remote Read Access requires the L_Key to have an accompanying R_Key.
  - Enable Remote Atomic Operation Access (If Atomic Ops supported).

    Remote Atomic Operation Access requires Local Write Access and the L_Key to have an accompanying R_Key.
  - Enable Memory Window Binding.

    Note: Local Read Access is always implied.

- Following are the input modifiers specific to the Local Invalidate Work Request:
  - Local Invalidate Fence Indicator
  - For a Memory Region
    - L_Key
    - R_Key - should be supplied if R_Key was assigned when MR was allocated.
    - Memory Region Handle
  - For a Memory Window
    - R_Key
    - Memory Window Handle

Output Modifiers:

- Number of WQEs successfully posted to the SQ. If verb result is operation completed successfully, then all WQEs were posted. Otherwise the verb result refers to the WQE that experienced the first WR Immediate Error and was not posted.
- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid QP handle.
  - Too many Work Requests posted.
  - Invalid operation type.
  - Invalid QP state.

    Note: This error is returned only when the QP is in the Reset, Init, or RTR states. It is not returned when the QP is in the Error or Send Queue Error states due to race conditions that could result in indeterminate behavior. Work Requests posted to the Send Queue while the QP is in the Error or Send Queue Error states are completed with a flush error.
  - Invalid MW Type. Consumer attempted to perform either: a Bind or Invalidate operation on a Type 1 Memory Window; or a zero length Bind operation on a Type 2 Memory Window
  - Invalid Scatter/Gather list format.
  - Invalid Scatter/Gather list length.
  - Atomic operations not supported.
  - Invalid address handle.
  - HCA doesn't support Base Memory Management Extensions.
  - HCA doesn't support Base Queue Management Extensions.
  - HCA doesn't support Local Invalidate Fencing.
  - HCA does not support Block Type Physical Buffers or was not opened in this mode.
  - HCA was not opened in Page mode.
  - HCA doesn't support Zero Based Virtual Address (ZBVA).

11.4.1.2 POST RECEIVE REQUEST

Description:

Builds one or more WQEs for the Receive Queue in the specified QP or the SRQ from the information contained in the list of Work Request submitted by the Consumer. These WQEs are added to the end of the Receive Queue or the SRQ, and the HCA is notified that one or more WQEs are ready to be processed.

Note: the Consumer can post up to the remaining capacity of a WQ without encountering a Too Many WRs Posted Immediate Error. If the Consumer posts more than the remaining capacity, an Immediate Error may be returned.

Control returns to the Consumer immediately after the WQEs have been submitted to the Receive Queue or the SRQ and the HCA has been notified that one or more WQEs are ready to process. When control returns, the list of Work Requests is in the scope of the Consumer and will no longer be modified or accessed below the Channel Interface.

C11-27: If the CI does not support the Base Queue Management Extensions, the CI shall return control to the Consumer immediately after the Work Request has been submitted to the Receive Queue.

C11-27.2.1: If the CI supports the Base Queue Management Extensions, the CI **shall** return control to the Consumer immediately after the list of Work Request has been submitted to the Receive Queue.

o11-5.2.3: If the HCA supports SRQ, the CI **shall** return control to the Consumer immediately after the list of Work Request has been submitted to the SRQ.

If an immediate error is returned, the QP state or SRQ state shall not be affected.

Input Modifiers:

- HCA handle.
- QP handle. Used when posting to a QP's Receive Queue and the QP is not associated with an SRQ.
- SRQ handle. Used when posting to an SRQ.
- A list of Work Requests. Where each entry consists a Work Request containing the information required to perform the Work Request. The WR modifiers that must be specified are dependent on the operation type specified. Each Work Request is defined as follows:
  - A user defined 64-bit Work Request ID.
  - Operation type. The only valid operation for the Receive Queue is the Receive operation.
  - Scatter/Gather list. This list can contain zero or more Data Segments.

    Note that for UD QPs, the first 40 bytes of the buffer(s) referred to by the Scatter/Gather list will contain the GRH of the incoming message. If no GRH is present, the contents of first 40

bytes of the buffer(s) will be undefined. The presence of the GRH will be indicated by a bit in the Work Completion.

Note that for Raw IPv6 QPs, the buffer(s) referred to by the Scatter/Gather list will contain the IPv6 header(s) of the incoming message followed by the message payload. The first 40 bytes will be used for the IPv6 routing header. Other IPv6 headers may follow the routing header before the message payload.

- Number of Data Segments in the scatter/gather list.

Output Modifiers:

- Number of WQEs successfully posted to the RQ or SRQ. If verb result is operation completed successfully, then all WQEs were posted. Otherwise the verb result refers to the WQE that experienced the first WR Immediate Error and was not posted.
- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid QP handle.
  - Too many Work Requests posted.
  - Invalid operation type.
  - Invalid QP state.
  - Invalid Scatter/Gather list format.
  - Invalid Scatter/Gather list length.
  - HCA doesn't support Base Queue Management Extensions.
  - Invalid SRQ handle.
  - QP Handle used on a QP that is associated with an SRQ.

11.4.2 COMPLETION QUEUE OPERATIONS

11.4.2.1 POLL FOR COMPLETION

Description:

Polls the specified CQ for a Work Completion. A Work Completion indicates that a Work Request for a Work Queue associated with the CQ is done.

If an entry is present, the Work Completion at the head of the CQ is returned to the Consumer.

If an immediate error, associated with executing the Poll CQ verb itself, is returned, the CQ and QP state shall not be affected.

The following table defines, classifies and associates wire level protocol NAK codes with completion errors that are possible on Work Requests posted to the Send Queue. Completion errors are returned through the completion queue as work completions.

C11-28: This compliance statement has been obsoleted.

C11-28.2.1: The CI **shall** return completion errors for a Work Request in the associated Work Completion for errors described in Table 95 Completion Error Types for Send Queues .

Table 95 Completion Error Types for Send Queues

Error Type	Completion Type	Transport Errors returned by responder (RC)	Transport Errors sent by responder (RD)
Bad Response	Processing	N/A	N/A
Invalid (local) EE Context Number	Processing	N/A	N/A
Invalid (local) EE Context State	Processing	N/A	N/A
Local EE Context Operation	Processing	N/A	N/A
Local Length	Interface	N/A	N/A
Local Length	Processing	N/A	N/A
Local Protection	Interface	N/A	N/A
Local Protection	Processing	N/A	Possibly a NAK - Invalid Request
Local QP Operation	Interface	N/A	N/A
Local QP Operation	Processing	N/A	Possibly a NAK - Invalid Request
Local RDD Violation	Processing	N/A	N/A
Memory Management Operation	Interface	N/A	N/A
Remote Access	Processing	NAK - Remote Access Violation	NAK - Remote Access Violation
Remote Invalid RD Request	Processing	N/A	NAK - Invalid RD Request
Remote Invalid Request	Processing	NAK - Invalid Request	NAK - Invalid Request
Remote Operation	Processing	NAK - Remote Operational Error	NAK - Remote Operational Error
RNR Retry Counter Exceeded	Processing	N/A	N/A
Transport Retry Counter Exceeded	Processing	Possibly a NAK - Sequence Error	Possibly a NAK - Sequence Error
Work Request Flushed	Processing	N/A	N/A

A Remote Q_Key violation and a Remote RDD Mismatch will both result in an Invalid RD Request completion error type for the requester's WQE. Since the same NAK code is returned in both cases, it is not possible for the requester to distinguish between them.

The following table defines, classifies and associates wire level protocol NAK codes with completion errors that are possible on Work Requests posted to the Receive Queue or Shared Receive Queue. Completion errors are returned through the completion queue as work completions.

C11-29: This compliance statement has been obsoleted.

C11-29.2.1: The CI **shall** generate the completion errors based on the NAK codes as shown in Table 96 Completion Error Types for RQs or SRQs .

Table 96 Completion Error Types for RQs or SRQs

Error Type	Completion Type	Transport Errors sent to Requester (RC)	Transport Errors sent to Requester (RD)
Invalid EE Context State	Processing	N/A	NAK - Invalid RD Request
Local Access	Processing	NAK - Remote Access Violation	NAK - Remote Access Violation
Local EE Context Operation	Processing	N/A	NAK - Invalid RD Request
Local Length	Processing	NAK - Remote Operational Error	NAK - Remote Operational Error
Local Protection	Processing	NAK - Remote Operational Error	NAK - Remote Operational Error
Local QP Operation	Processing	NAK - Remote Operational Error	NAK - Remote Operational Error
Aborted	Processing	N/A	Possibly RNR NAK
Remote Invalid Request	Processing	NAK - Invalid Request	NAK - Invalid Request
Work Request Flushed	Processing	N/A	N/A

Input Modifiers:

- HCA handle.
- CQ handle.

Output Modifiers:

- The Work Completion containing information relating to the completed Work Request if an entry is present on the CQ. If the status of the operation that generates the Work Completion is anything other than success, the contents of the Work Completion are undefined except as noted below. The contents of a Work Completion are:
  - The 64-bit Work Request ID set by the Consumer in the associated Work Request. This is always valid, regardless of the status of the operation.
  - The operation type specified in the completed Work Request.
  - The valid operation types are:

- Send (for WRs posted to the Send Queue)
- RDMA Write (for WRs posted to the Send Queue)
- RDMA Read (for WRs posted to the Send Queue)
- Compare and Swap (for WRs posted to the Send Queue)
- Fetch and Add (for WRs posted to the Send Queue)
- Fast Register Physical MR (for WRs posted to the Send Queue)
- Local Invalidate (for WRs posted to the Send Queue)
- Memory Window Bind (for WRs posted to the Send Queue)
- Send Data Received (for WRs posted to the Receive Queue or Shared Receive Queue)
- RDMA with Immediate Data Received (for WRs posted to the Receive Queue or Shared Receive Queue)

- The number of bytes transferred.

  The number of bytes transferred is returned in Work Completions for Receive Work Requests for incoming Sends and RDMA Writes with Immediate Data. This does not include the length of any immediate data.

  The number of bytes transferred is returned in Work Completions for Send Work Requests for RDMA Read and Atomic Operations.

  For the RQ of a UD QP that is not associated with an SRQ or for an SRQ that is associated with a UD QP, the number of bytes transferred is the payload of the message plus the 40 bytes reserved for the GRH. For the RQ of a UD QP that is not associated with an SRQ or for an SRQ that is associated with a UD QP, the 40 bytes is always included, whether or not the GRH is present.
- R_Key Invalidated Indicator. If set, Received Message Invalidated an R_Key. Note, if an incoming Send with Invalidate completed with a Memory Management Operation Error, then the R_Key was not Invalidated.
  - Invalidated R_Key. The R_Key invalidated by the Received Message.
- QP Number. Set if the Base Queue Management Extension or Shared Receive Queue Extension is supported.
- Immediate data indicator. This is set if immediate data is present.
- 4-byte immediate data.

- Remote node address and QP. Returned only for Datagram services. The address information returned for incoming Datagrams is shown in Table 97 Datagram addressing information .

Table 97 Datagram addressing information

Reliable Datagrams	Unreliable Datagrams	Raw IPv6	Raw Ethertype
16-bit SLID	16-bit SLID	16-bit SLID	16-bit SLID
4-bit SL	4-bit SL	4-bit SL	4-bit SL
24-bit Source QP	24-bit Source QP		16-bit Ethertype
24-bit local EE Number	DLID Path Bits[a]	DLID Path Bits[a]	DLID Path Bits[a]

a. Note: Not applicable for Multicast messages

  - GRH Present indicator, for UD RQs only. If this indicator is set, the first 40 bytes of the buffer(s) referred to by the Scatter/Gather list will contain the GRH of the incoming message. If it is not set, the contents of first 40 bytes of the buffer(s) will be undefined. Contents of the payload of the message will begin after the first 40 bytes
  - P_Key index, for GSI only.
  - Status of the operation. This is always valid.
    - These status codes are covered in Completion Return Status, with NAK codes reported according to Completion Error Types for Send Queues and Completion Error Types for RQs or SRQs.
  - Freed Resource Count (see 10.8.5.1 Freed Resource Count on page 520). This is always valid, regardless of the status of the operation.
- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid CQ handle.
  - CQ empty.

11.4.2.2 REQUEST COMPLETION NOTIFICATION

Description:

Requests the CQ event handler be called when the next completion entry of the specified type is added to the specified CQ. The handler is called at most once per Request Completion Notification call for a

particular CQ. Any CQ entries that existed before the notify is enabled will not result in a call to the handler.

Completion Events are one of two types: solicited or unsolicited. A Solicited Completion Event occurs when an incoming Send or RDMA Write with Immediate Data message, with the Solicited Event header bit set causes a successful Receive Work Completion to be added to a CQ; or, when an unsuccessful Work Completion is added to a CQ. An Unsolicited Completion Event occurs when any other successful Receive Work Completion, or any successful Send Work Completion, is added to a CQ.

C11-30: The CI **shall** support both solicited and unsolicited Completion Event Types.

When the Consumer requests completion notification, it must specify whether the notification callback is invoked for either:

- the next Solicited Completion Event only, or
- the next Solicited or Unsolicited Completion Event.

C11-30.1.1: When “next Solicited Completion Event only” is outstanding, the CI shall invoke a notification callback when any of the following conditions occur:

- An incoming Send with the Solicited Event Header bit set causes a successful Receive Work Completion to be added to the specified CQ.
- An incoming RDMA Write with Immediate Data with the Solicited Event Header bit set causes a successful Receive Work Completion to be added to the specified CQ.
- An unsuccessful Send or Receive Work Completion is added to the specified CQ.

C11-30.1.2: When “next Solicited or Unsolicited Completion Event” is outstanding, the CI shall invoke a notification callback when any Work Completion is added to the specified CQ.

If a Request Completion Notification is pending, subsequent calls to Request Completion Notification for the same CQ prior to the completion event affect only when the notification occurs. A Request Completion Notification for the next completion event takes precedence over a Request Completion Notification for a solicited event completion for the same CQ.

If multiple calls to Request Completion Notification have been made for the same CQ and at least one of the requests set the type to the next completion, the CQ event handler will be called when the next completion is added to that CQ. The CQ event handler will be called

only once, even though multiple CQ notification requests were made prior to the completion event for the specified CQ.

Once the CQ event handler is called, another completion notification request must be registered before the CQ event handler will be called again.

C11-31: When a completion notification request is outstanding on a CQ for a *solicited* completion type and another request for that CQ is made that specifies a notification for the *next* completion, the CI **shall** change the outstanding completion notification type to the *next* completion.

C11-32: When a completion notification request is outstanding on a CQ for the *next* completion and another notification request for that CQ is made, the CI **shall not** change the outstanding completion notification type.

A CQ event handler must be specified prior to calling this routine. If the CQ event handler has not been registered when the event is generated, the handler call will not be made.

When the CQ event handler is called, it only indicates a new entry was added to the specified CQ. The HCA and CQ handles are passed to the CQ event handler so the CQ event handler can determine which CQ caused it to be called.

Once the handler routine has been invoked, the Consumer must call Request Completion Notification again to be notified when a new entry is added to that CQ.

It is the responsibility of the Consumer to call the Poll for Completion Verb to retrieve a Work Completion.

Note: If the Consumer Requests Completion Notification on a CQ Handle that does not have a CQ Event Handler ID associated with the CQ, the operation will have no effect. That is, no completion event will be generated.

Input Modifiers:

- HCA handle.
- CQ handle.
- Type of completion notification requested. The type is either the next completion or when a solicited completion occurs.

Output Modifiers:

- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid CQ handle.

- Invalid completion notification type.

11.5 EVENT HANDLING

11.5.1 SET COMPLETION EVENT HANDLER

Description:

Associates a Completion Handler Identifier with a Completion Event Handler Address. If the HCA supports the Base Queue Management Extensions, more than one CQ event handler can be registered per HCA.

For a given Completion Handler Identifier, additional calls to this Verb will overwrite the Completion Event Handler Address associated with the Completion Handler Identifier.

This call does not automatically request a notification on a completion event. The Request Completion Notification Verb must be called in order to request notification.

The parameters passed to the CQ event handler are:

- HCA handle.
- CQ handle.

Input Modifiers:

- HCA handle.
- Completion Event Handler Address.
- Completion Event Handler Identifier:
  - If zero, the CI will create a Completion Handler Identifier and the Completion Event Handler Address will be assigned.
  - If non-zero, the CI will replace the Completion Event Handler Address associated with the existing Completion Handler identified by the Completion Event Handler Identifier. If the Completion Event Handler Address is zero, then the Completion Event Handler Address is cleared. Note: A completion event must not be generated when the CQ is associated with a cleared Completion Event Handler.

Output Modifiers:

- Completion Event Handler Identifier. Returned only if the input modifier “Completion Event Handler Identifier” is set to zero.
- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid Completion Event Handler Identifier.

- HCA doesn't support Base Queue Management Extensions.
- Insufficient resources to complete request.

11.5.2 SET ASYNCHRONOUS EVENT HANDLER

Description:

Registers the asynchronous event handler. Only one asynchronous event handler can be registered per HCA. Additional calls to this Verb will overwrite the handler routine to be called. Additional calls will not generate an additional handler routine.

C11-33: The CI **shall** use the asynchronous event handler specified in this Verb even in the case where an existing asynchronous event handler has already been registered.

After the asynchronous event handler is registered, all subsequent asynchronous events will result in a call to the handler. Until an asynchronous event handler is registered, asynchronous events will be lost.

The parameters passed to the asynchronous event handler when it is invoked are:

- HCA handle.
- Event record. This contains information which indicates the resource type and identifier as well as which event occurred. See Asynchronous Events for more information.

Input Modifiers:

- HCA handle.
- Handler address.

Output Modifiers:

- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.

11.6 RESULT TYPES

11.6.1 IMMEDIATE RETURN RESULTS

This section contains a list of the possible Verb return results. All results except "Operation completed successfully" are due to interface errors in the Immediate Error category. Not all Verbs return all results.

Successful return result:

- Operation completed successfully.

Resource errors:

- Insufficient resources to complete request.
- CQ capacity requested exceeds HCA capability.
- Maximum number of Work Requests requested exceeds HCA capability.
- Maximum number of scatter/gather elements requested exceeds HCA capability.
- Too many Work Requests posted.
- Number of available Raw Datagram QPs exceeded.
- Number of QPs attached to multicast groups exceeded.
- HCA already in use.

HCA attribute errors:

- Invalid HCA name.
- Invalid HCA handle.
- MTUCap of HCA port exceeded.
- Invalid Port.
- Invalid Counter specified.

Address errors:

- Invalid Address handle.

QP errors:

- Invalid QP handle.
- Cannot change QP attribute.
- Invalid QP state.
- Invalid Service Type for this QP.
- QP is already in use.
- Atomic operations not supported.
- Raw Datagrams not supported.
- Reliable Datagrams not supported.
- Invalid operation type.
- Invalid Scatter/Gather list format.
- Invalid Scatter/Gather list length.
- Invalid path migration state.
- Invalid Special QP type.
- Invalid Address Handle
- More outstanding entries on WQ than size specified.

- The QP is still attached to one or more multicast groups.

CQ errors:

- Invalid CQ handle.
- More entries outstanding on CQ than capacity specified.
- One or more Work Queues still associated with the CQ.
- CQ empty.
- Invalid completion notification type.
- CQ has overrun or has become inaccessible.

EE Context errors:

- Invalid EE Context handle.
- Invalid EE Context state.
- Cannot change EE Context attribute.

QP or EE Context errors:

- Invalid path migration state.
- Reliable Datagram Domain is in use.
- Invalid Reliable Datagram Domain.
- Invalid RNR NAK Timer Field value.

Memory operation errors:

- Invalid Protection Domain.
- Protection Domain is in use.
- Invalid Virtual Address.
- Invalid Length.
- Invalid Physical Buffer List entry.
- Invalid Offset.
- Invalid L_Key.
- Invalid R_Key.
- Invalid Memory Region handle.
- Invalid Memory Window handle.
- Invalid Access Control specifier.
- Operation denied; Region still has bound Window(s)

Multicast errors:

- Invalid multicast MLID.
- Invalid Multicast group MGID.

Partition table errors:

- P_Key index out of range.

- P_Key index specifies invalid entry in the P_Key table.

Verb Extension related errors:

- HCA does not support Block Type Physical Buffers or was not opened in this mode.
- HCA was not opened in Page mode.
- HCA does not support Base Memory Management Extensions.
- HCA does not support Base Queue Management Extensions.
- HCA does not support Local Invalidate Fencing.
- HCA does not support multiple PB sizes per MR.
- HCA does not support SRQ.
- HCA does not support resizing SRQ.
- HCA does not support ZBVA.
- QP Handle used on a QP that is associated with an SRQ.
- QP still has Type 2A MW bound to it.
- Invalid Completion Event Handler ID.
- Invalid MW Type.
- Invalid SRQ handle.
- SRQ is in the Error State.
- SRQ Limit exceeds maximum number of Work Requests allowed on the SRQ.
- SRQ still has QPs associated with it.

11.6.2 COMPLETION RETURN STATUS

Describes the possible Work Completion status error return results. These are errors that occur during the processing of a Work Request and can be reported in the Work Completion status.

- Success - Operation completed successfully.
- Local Length Error - Generated for a Work Request posted to the local Send Queue when the sum of the Data Segment lengths exceeds the message length for the channel adapter port. Generated for a Work Request posted to the local Receive Queue when the sum of the Data Segment lengths is too small to receive a valid incoming message or the length of the incoming message is greater than the maximum message size supported by the HCA port that received the message.
- Local QP Operation Error - An internal QP consistency error was detected while processing this Work Request.

- Local EE Context Operation Error - An internal EE Context consistency error was detected while processing this Work Request.
- Local Protection Error - The locally posted Work Request's Data Segment does not reference a Memory Region that is valid for the requested operation.
  - On a Reserved L_Key:
    - The Verbs Consumer does not have Reserved L_Key access enabled.
- Work Request Flushed Error - A Work Request was in process or outstanding when the QP transitioned into the Error State.
- Memory Management Operation Error - The Verbs Consumer had insufficient rights to perform the operation, because:
  - On a Fast-Register this includes:
    - The Verbs Consumer does not have Fast Register access enabled;
    - Requested PBL exceeds size of allocated PBL;
    - Remote access was requested, but L_Key does not have an accompanying R_Key; or
    - Memory access was attempted on an L_Key or R_Key that is in the Invalid State.
  - On a Memory Window this includes:
    - The Verbs Consumer has insufficient access rights;
    - A zero length Bind Memory Window or Post Send Bind WR operation on a Type 2 Memory Window
    - Verb Consumer attempted to Bind a MW to a Zero Based Virtual Address Memory Region.
  - On an Invalidate Operation this includes:
    - Memory access was attempted on an L_Key or R_Key that is in the Invalid State;
    - Memory Region could not be Invalidated, because it is a Shared Memory Region;
    - Memory Region can not be invalidated because it has bound Memory Window; or
    - Memory Region could not be Invalidated, because it was created through a Register Memory Region or Reregister Memory Region.
    - Memory Window could not be Invalidated, because it was a Type 1 Memory Window.

The following errors are reported only for Reliable QPs.

- Bad Response Error - An unexpected transport layer opcode was returned by the responder.
- Local Access Error - A protection error occurred on a local data buffer during the processing of a RDMA Write with Immediate Data operation sent from the remote node.
- Remote Invalid Request Error - The responder detected an invalid message on the channel. Possible causes include the operation is not supported by this receive queue, insufficient buffering to receive a new RDMA or Atomic Operation request, or the length specified in an RDMA request is greater than 2^{31} bytes.

  In the case where the buffer size is insufficient to handle the request, the number of bytes transferred into the buffer is indeterminate. However, the CI shall not write beyond the buffer bounds.
- Remote Access Error - A protection error occurred on a remote data buffer to be read by an RDMA Read, written by an RDMA Write or accessed by an atomic operation. This error is reported only on RDMA operations or atomic operations.
- Remote Operation Error - The operation could not be completed successfully by the responder. Possible causes include a responder QP related error that prevented the responder from completing the request or a malformed WQE on the Receive Queue.
- Transport Retry Counter Exceeded - The local transport timeout retry counter was exceeded while trying to send this message.
- RNR Retry Counter Exceeded - The RNR NAK retry count was exceeded.

The following errors are reported only for RD QPs or EE Contexts.

- Local RDD Violation Error - The RDD associated with the QP does not match the RDD associated with the EE Context
- Remote Invalid RD Request - The responder detected an invalid incoming RD message. Causes include a Q_Key or RDD violation.
- Aborted Error - The operation was aborted:
  - For RD, the requester aborted the operation. One possible cause is the requester suspended the operation and will retry it later using a new Receive WQE. The other possible cause is the requester abandoned the operation and placed the requester QP in the SQEr state.
  - For UD QPs associated with an SRQ, the responder aborted the operation.
- Invalid EE Context Number - An invalid EE Context number was detected.

- Invalid EE Context State - Operation is not legal for the specified EE Context state.

All the above completion errors are applicable for WRs submitted through the Post Send Request and Post Receive Request verbs, except for the Memory Window Bind Error status. WR submitted through the Bind Memory Window verb must complete with one of the following error codes: Success, Memory Window Bind Error, Local QP Operation Error or Work Request Flushed Error.

11.6.3 ASYNCHRONOUS EVENTS

This section describes the asynchronous events. Asynchronous events are separated into four categories: Affiliated asynchronous events, Affiliated asynchronous errors, Unaffiliated asynchronous events and Unaffiliated asynchronous errors. Both kinds of asynchronous errors are defined in 10.10.2.3 Asynchronous Errors on page 531.

Affiliated asynchronous events have been separated into two categories because the behavior of the QP/EE Context when the events occur are different.

C11-34: When an affiliated asynchronous error occurs, the CI **shall** cause the QP/EE to transition to the Error state.

C11-35: When an affiliated asynchronous event occurs, the CI **must** leave the QP/EE in the QP/EE State that it was in when the asynchronous event occurred.

Unaffiliated asynchronous errors are those which cannot be associated with a specific QP or EE Context.

The Verbs Consumer must register a handler as described in Set Asynchronous Event Handler to be notified that an asynchronous event has occurred. This mechanism is used to collect information about both events and the errors.

11.6.3.1 AFFILIATED ASYNCHRONOUS EVENTS

Affiliated asynchronous events are advisories to the Verb Consumer that the specified event has occurred on the specified QP or EE Context. Events in this category are not considered to be errors by the Channel Interface, so the QP/EE state remains unchanged.

- Path Migrated - Indicates the connection has migrated to the alternate path.
- Communication Established - Indicates the first packet has arrived for the Receive Work Queue where the QP/EE is still in the RTR state. The handle of the QP/EE, which was the destination of this packet is

returned in the event record. This event may be used by the Communication Manager as shown in the state diagram in 12.9.6 Communication Establishment - Passive on page 688 and described in 12.9.7.2 Passive States. The Communication Manager may receive this event while it is already in the Established state; this is not an error.

C11-36: For RC and UC service, the CI **shall** generate a Communication Established Affiliated Asynchronous Event if the first packet arrives while the QP is still in the RTR state, and the first packet is processed with no errors. The CI is allowed to generate this Affiliated Asynchronous Event under some error conditions, but shall not generate the event unless inbound packet validation is error-free at least past the “actual PSN=ePSN?” box (see Figure 88, Figure 112).

o11-5.1.1: If the CI supports RD service, the CI **shall** generate a Communication Established Affiliated Asynchronous Event if the first packet arrives while the EEC is still in the RTR state, and the first packet is processed with no errors. The CI is allowed to generate this Affiliated Asynchronous Event under some error conditions, but shall not generate the event unless inbound packet validation is error-free at least past the “actual PSN=ePSN?) box (see Figure 89).

For UD and Raw service types, generation of the Communication Established Affiliated Asynchronous Event is allowed, but is strongly discouraged.

- Send Queue Drained - Indicates that the Send Queue of the specified Queue Pair or EE has completed the outstanding Messages in progress when the state change was requested and, if applicable, has received all acknowledgements for those messages; The event is also generated if the transition to SQD has been aborted by a transition into SQError, Error or Reset state.

o11-5.2.4: If the HCA supports SRQ, for RC and UD service, the CI shall generate a SRQ Limit Reached Affiliated Asynchronous Event if SRQ Limit Event generation mechanism is armed and the SRQ Limit is reached. The SRQ Limit is reached whenever the number of SRQ WQEs is less than the SRQ Limit.

The CI shall reset the SRQ Limit to zero when the event has been generated.

o11-5.2.5: If the HCA supports SRQ, for RC and UD service, the CI shall generate a Last WQE Reached Affiliated Asynchronous Event on a QP that is in the Error State and is associated with an SRQ when either:

- a CQE is generated for the last WQE, or

- the QP gets in the Error State and there are no more WQEs on the RQ.

o11-5.2.6: If the HCA supports SRQ, for RC and UD service, the CI shall not generate a Last WQE Reached Affiliated Asynchronous Event on a QP after the CI surfaced a Local Work Queue Catastrophic Error on the same QP.

If the HCA experiences an Local Work Queue Catastrophic Error and the Last WQE Reached Affiliated Asynchronous Event does not occur, then the Consumer must destroy all QPs and the SRQ to reclaim all the WQEs associated with that QP.

11.6.3.2 AFFILIATED ASYNCHRONOUS ERRORS

- CQ Error - Indicates an error occurred when writing an entry to the Completion Queue.

C11-37: The CI **shall** generate a CQ Error when an error, other than CQ overrun, occurs while writing an entry to the CQ.

C11-38: The CI **shall** generate a CQ Error when a CQ overrun is detected.

This condition will result in an Affiliated Asynchronous Error for any associated Work Queues when they attempt to use that CQ. Completions can no longer be added to the CQ. It is not guaranteed that completions present in the CQ at the time the error occurred can be retrieved. Possible causes include a CQ overrun or a CQ protection error.

- Local Work Queue Catastrophic Error - An error occurred while accessing or processing the Work Queue that prevents reporting of completions.

C11-39: The CI **shall** generate a Local Work Queue Catastrophic Error when a Work Queue associated with a CQ that caused the CQ Error to be generated attempts to use that CQ.

C11-40: The CI **shall** generate a Local Work Queue Catastrophic Error when an error occurred while accessing or processing the Work Queue that prevents reporting of completions.

- Invalid Request Local Work Queue Error - The transport layer detected a transport OpCode violation at the Responder. Possible causes are: Unsupported or Reserved OpCode; or Out of Sequence OpCode.

C11-40.1.1: For RC Service, the CI **shall** generate an Invalid Request Local Work Queue Error when the transport layer detects a transport Op-

Code violation at the Responder. The Responder's affiliated QP **shall be** placed in the error state.

- Local Access Violation Work Queue Error - The transport layer detected a Request access violation at the Responder. Possible causes are: Misaligned Atomic; Too many RDMA Read or Atomic Requests; R_Key violation; or length errors without immediate data.

C11-40.1.2: For RC Service, the CI **shall** generate a Local Access Violation Work Queue Error when the transport layer detects a Request access violation at the Responder. The Responder's affiliated QP **shall be** placed in the error state.

- Local EE Context Catastrophic Error - An Error occurred while accessing or processing the EE Context that prevents reporting of completions.

o11-5.a1: If the CI supports RD Service, the CI **shall** generate a EE Context Catastrophic Error when an error occurred while accessing or processing the EE Context that prevents reporting of completions.

- Path Migration Request Error - Indicates the incoming path migration request to this QP/EE was not accepted. The validation process is defined in section Migration Request.

o11-6: If the CI supports automatic path migration, the CI **shall** generate a Path Migration Request Error when the incoming path migration request to this QP/EE was not accepted.

- SRQ Catastrophic Error - An error occurred while processing or accessing the SRQ that prevents dequeuing a WQE from the SRQ and reporting of receive completions.

o11-6.2.1: If the HCA supports SRQ, the CI shall generate an SRQ Catastrophic Error, if an error, other than “Resource Not Ready”, prevents the HCA from dequeuing a WQE from the SRQ. When this error occurs the CI shall place the SRQ in the Error state.

o11-6.2.2: If the HCA supports SRQ, for a QP that is associated with an SRQ the CI shall generate a Local Work Queue Catastrophic Error and place the QP in the Error state, if either:

- an error, other than “Resource Not Ready”, prevents the HCA from dequeuing a WQE from the SRQ; or
- a QP tries to dequeue a WQE from the SRQ, but the associated SRQ is already in the Error State.

11.6.3.3 UNAFFILIATED ASYNCHRONOUS EVENTS

- Port Active - issued when the link becomes active.

o11-6.1.1: If the Port Active event is supported, the CI shall generate a Port Active event when the link is declared active.

Using the definitions for “available” and “unavailable” states in the Port Error description below, the “Port Active” event is generated when the link associated with an HCA port transitions from an unavailable to an available state.

- Client Reregistration Event - issued when SM requests client reregistration (see 14.4.11 Client Reregistration on page 881).

o11-6.2.3: If the CI indicates that the port supports client reregistration, the CI shall generate a Client Reregistration Event when the SMA receives this request from the SM.

11.6.3.4 UNAFFILIATED ASYNCHRONOUS ERRORS

- Local Catastrophic Error - An error occurred which cannot be attributable to any resource and CI behavior is indeterminate.

C11-41: The CI **shall** generate a Local Catastrophic Error when an error occurred which cannot be attributable to any resource and CI behavior is indeterminate.

- Port Error - Issued when the link is declared unavailable.

C11-42: The CI **shall** generate a Port Error when the link is declared unavailable. Port Errors **shall** have no effect on QP/EE State.

Using the definitions of Link States, the “unavailable” states are considered to be: Down, Initialize and Armed. The “available” states are Active and ActDefer. The “Port Error” unaffiliated asynchronous error is generated when the link associated with an HCA port transitions from an available to an unavailable state.

11.6.4 VERB EXTENSION SUMMARY

Table 98 List of Extended Verbs and Optional Modifiers lists all the verbs, modifiers, and verb results which must be supported by an implementation that supports any of the verb extensions in this specification. If an implementation does not support any of these verb extensions, then that

implementation will not support any of these new verbs, modifiers, and verb results.

Table 98 List of Extended Verbs and Optional Modifiers

Verb	1.1 Spec[a]	Ext. Verbs Group[b]	New Modifiers[c]
Open HCA	M	BL	Added: I: The type of Physical Buffer that will be used on the HCA. R: Block type Physical Buffers are not supported.
Query HCA	M	VE	Added: O: The ability of this HCA to support multiple Physical Buffer sizes per Memory Region.
		BMM	Added: O: Ability of this HCA to support the Base Memory Management Extensions. O: Value of Reserved L_Key. O:Maximum Physical Buffer List size supported by this HCA when invoking the Allocate L_Key verb. O: List of Page sizes supported by this HCA. O: Bound Type 2 Memory Window Association mechanism O: Ability of this HCA to support multiple physical buffer sizes per Memory Region.
		BL	Added: O: Ability of this HCA to support Block List Physical Buffer Lists. O: Range of Block sizes supported by this HCA.
		ZBVA	Added: O: Ability of this HCA to support Zero Based Virtual Addresses.
		LIF	Added: O: Ability of this HCA to support Local Invalidate Fencing.
		BQM	Added: O: Ability of this HCA to support the Base Queue Management Extensions. O: Maximum number of CQ Event Handlers.
		SRQ	Added: O: Ability of this HCA to support the Shared Receive Queues. O: Maximum number of SRQs. O: Maximum number of WRs per SRQ. O: Maximum number of Scatter/Gather entries per SRQ WR. O: Ability to modify the maximum number of WRs per SRQ.
		Rereg	Added: O: Client reregistration supported

Table 98 List of Extended Verbs and Optional Modifiers (Continued)

Verb	1.1 Spec[a]	Ext. Verbs Group[b]	New Modifiers[c]
Modify HCA Attributes	M	NA	
Close HCA	M	NA	
Allocate Protection Domain	M	NA	
Deallocate Protection Domain	M	NA	
Allocate Reliable Datagram Domain	R	NA	
Deallocate Reliable Datagram Domain	R	NA	
Create Address Handle	M	NA	
Modify Address Handle	M	NA	
Query Address Handle	M	NA	
Destroy Address Handle	M	NA	
Create Shared Receive Queue	N	SRQ	All
Modify Shared Receive Queue	N	SRQ	All
Query Shared Receive Queue	N	SRQ	All
Destroy Shared Receive Queue	N	SRQ	All
Create QP	M	SRQ	Added: I: SRQ Handle, if QP is to be associated to an SRQ. R: Invalid SRQ handle. Became optional: I: The maximum number of outstanding Work Requests the Consumer expects to submit to the Receive Queue. I: The maximum number of scatter/gather elements the Consumer will specify in a Work Request submitted to the Receive Queue. O: The actual number of outstanding Work Requests supported on the Receive Queue. O: The actual number of scatter/gather elements that can be specified in Work Requests submitted to the Receive Queue.
		BMM	Added: I: Enable or disable Fast Register PMR and Reserved L_Key operations.
		VE	Added: R: HCA doesn't support Base Memory Management Extensions. R: HCA does not support SRQ.

Table 98 List of Extended Verbs and Optional Modifiers (Continued)

Verb	1.1 Spec[a]	Ext. Verbs Group[b]	New Modifiers[c]
Modify QP	M	SRQ	Became optional: I: The maximum number of outstanding Work Requests the Consumer expects to submit to the Receive Queue. O: The actual number of outstanding Work Requests supported on the Receive Queue.
		BMM	Added: R: QP still has Type 2A MWs bound to it.
Query QP	M	SRQ	Added: O: SRQ Handle Became optional: O: The actual number of outstanding Work Requests supported on the Receive Queue. O: The actual number of scatter/gather elements that can be specified in Work Requests submitted to the Receive Queue.
		BMM	Added: O: Fast Register PMR and Reserved L_Key operations enabled or disabled.
Destroy QP	M	BMM	Added: R: QP still has Type 2A MWs bound to it.
Get Special QP	M	NA	
Create Completion Queue	M	BQM	Added: I: Completion Event Handler Identifier. R: Invalid Completion Event Handler Identifier.
		VE	Added: R: HCA doesn't support Base Queue Management Extensions.
Query Completion Queue	M	BQM	Added: O: Completion Event Handler Identifier
Resize Completion Queue	M	NA	
Destroy Completion Queue	M	NA	
Create EE Context	R	NA	
Modify EE Context Attributes	R	NA	
Query EE Context	R	NA	
Destroy EE Context	R	NA	
Allocate L_Key	N	BMM	All

Table 98 List of Extended Verbs and Optional Modifiers (Continued)

Verb	1.1 Spec[a]	Ext. Verbs Group[b]	New Modifiers[c]
Register Memory Region	M	ZBVA	Added: I: Type of VA
		VE	Added: R: HCA doesn't support ZBVA
Register Physical Memory Region	M	BMM	Added: I: Key ownership requested I: Key to use on the new L_Key and R_Key. O: Actual size of the PBL resources allocated.
		ZBVA	Added: I: Type of VA Became optional: I: VA requested by the Consumer for the first byte of the region
		BL	Added: I: Physical Buffer Type Changed: I: Page size - used for lists with multiple page sizes I: Physical buffer size (The input modifier became per physical buffer) - used for lists containing buffers and same sized pages
		VE	Added: R: HCA doesn't support Base Memory Management Extensions. R: HCA doesn't support ZBVA. R: HCA doesn't support Block Type Physical Buffers or was not opened in this mode. R: HCA doesn't support multiple PB sizes per MR.
Query Memory Region	M	BMM	Added: O: L_Key state. O: Ownership attributes for the key portion of L_Key and R_Key. O: Sharing attributes of the Memory Region: Shared vs. Not Shared. O: Actual number of allocated Physical Buffer List entries.
		ZBVA	Added: O: Type of VA
Deregister Memory Region	M	NA	
Reregister Memory Region	M	ZBVA	Added: I: Type of VA
		VE	Added: R: HCA doesn't support ZBVA

Table 98 List of Extended Verbs and Optional Modifiers (Continued)

Verb	1.1 Spec[a]	Ext. Verbs Group[b]	New Modifiers[c]
Reregister Physical Memory Region	M	BMM	Added: I: Key to use on the new L_Key and R_Key. I: Key ownership requested I: L_Key Key I: R_Key Key O: Actual size of the PBL resources allocated
		ZBVA	Added: I: Type of VA Became optional: I: VA requested by the Consumer for the first byte of the region
		BL	Added: I: Physical Buffer Type Changed: I: Page size - used for lists with multiple page sizes I: Physical buffer size (The input modifier became per physical buffer) - used for lists containing buffers and same sized pages
		VE	Added: R: HCA doesn't support Base Memory Management Extensions. R: HCA doesn't support ZBVA. R: HCA doesn't support Block Type Physical Buffers or was not opened in this mode. R: HCA doesn't support multiple PB sizes per MR.
Register Shared Memory Region	M	ZBVA	Added: I: Type of VA
		VE	Added: R: HCA doesn't support ZBVA
Allocate Memory Window	M	BMM	Added: I: Type of Memory Window.
		VE	R: HCA doesn't support Type 2 Memory Windows.
Query Memory Window	M	BMM	Added: O: Type of Memory Window. O: R_Key state.
Bind Memory Window	M	BMM	Added: R: Invalid MW Type
Deallocate Memory Window	M	NA	
Attach QP to Multicast Group	U	NA	
Detach QP from Multicast Group	U	NA	

Table 98 List of Extended Verbs and Optional Modifiers (Continued)

Verb	1.1 Spec[a]	Ext. Verbs Group[b]	New Modifiers[c]
Post Send Request	M	BQM	Added: I: A list of Work Requests (1.1 required a single WR) O: Number of WQEs successfully posted to the SQ
		BMM	Changed: I: Operation type (Added: Fast Register Physical MR, Local Invalidate, Type 2 Memory Window Bind) Added: I: (Remote) Invalidate indicator. I: R_Key that is to be included in the Send with Invalidated. Added for Bind Type 2: I: New R_Key Key. I: Memory Window Handle. I: R_Key I: Memory Region Handle. I: L_Key I: Virtual Address I: Length of range to be bound in bytes. I: Access Control Added for Fast Register: I: Key to use on the new L_Key and R_Key. I: L_Key I: R_Key I: Memory Region Handle I: Physical Buffer List. I: Starting physical address of each physical buffer. I: Page size - used for lists with multiple page sizes I: Physical buffer size (The input modifier became per physical buffer) - used for lists containing buffers and same sized pages. I: Total number of Physical Buffers in the list. I: The VA requested by the Consumer for the first byte of the region. I: Length of Region to be registered in bytes. I: Offset of Region's starting IOVA within the first physical buffer. I: Access Control Added for Local Invalidate: I: L_Key I: R_Key I: Memory Region Handle I: Memory Window Handle R: Invalid MW Type

Table 98 List of Extended Verbs and Optional Modifiers (Continued)

Verb	1.1 Spec[a]	Ext. Verbs Group[b]	New Modifiers[c]
Post Send Request (cont'd)	M	ZBVA	Added for Bind Type 2: I: Type of VA. Added for Fast Register: I: Type of VA.
		LIF	Added for Local Invalidate: I: Local Invalidate Fence Indicator
		VE	Added: R: HCA doesn't support Base Memory Management Extensions. R: HCA doesn't support Base Queue Management Extensions. R: HCA doesn't support Local Invalidate Fencing. R: HCA does not support Block Type Physical Buffers or was not opened in this mode. R: HCA doesn't support Zero Based Virtual Address (ZBVA). R: QP Handle used on a QP that is associated with an SRQ.
Post Receive Request	M	BQM	Added: I: A list of Work Requests (1.1 required a single WR) O: Number of WQEs successfully posted to the RQ
		SRQ	Became Optional: I: QP handle Added: I: SRQ handle
		VE	Added: R: HCA doesn't support Base Queue Management Extensions. R: HCA doesn't support SRQ.
Poll for Completion	M	BQM or SRQ	Added: O: QP number
		BMM	Changed: O: The operation type (added: Fast Register Physical MR, Local Invalidate) Added: O: R_Key Invalidated Indicator. O: Invalidated R_Key.
Request Completion Notification	M	NA	

Table 98 List of Extended Verbs and Optional Modifiers (Continued)

Verb	1.1 Spec[a]	Ext. Verbs Group[b]	New Modifiers[c]
Set Completion Event Handler	M	BQM	Changed: I: Completion Event Handler Address (allowed to be zero) Added: I: Completion Handler Identifier R: Invalid Completion Event Handler Identifier. R: Insufficient resources to complete request.
		VE	Added: R: HCA doesn't support Base Queue Management Extensions.
Set Asynchronous Event Handler	M	NA	

a. 1.1 Spec Requirements - Legend:
- M - Mandatory
- N - Not Required
- R - RD Service
- U - UD Multicast

b. Extended Verbs Group - Legend:
- NA - Not Applicable
- VE - HCA supports any one of the Verb Extensions, but not the one being requested.
- BQM - Base Queue Management
- BMM - Base Memory Management
- SRQ - Shared Receive Queue
- BLBL - Block List Physical Buffer List
- ZBVA - Zero Based Virtual Address
- LIF - Local Invalidate Fencing
- Rereg - Client reregistration

c. New Modifiers - Legend:
- I - Input Modifier
- O - Output Modifier
- R - Verb Result

CHAPTER 12: COMMUNICATION MANAGEMENT

12.1 OVERVIEW

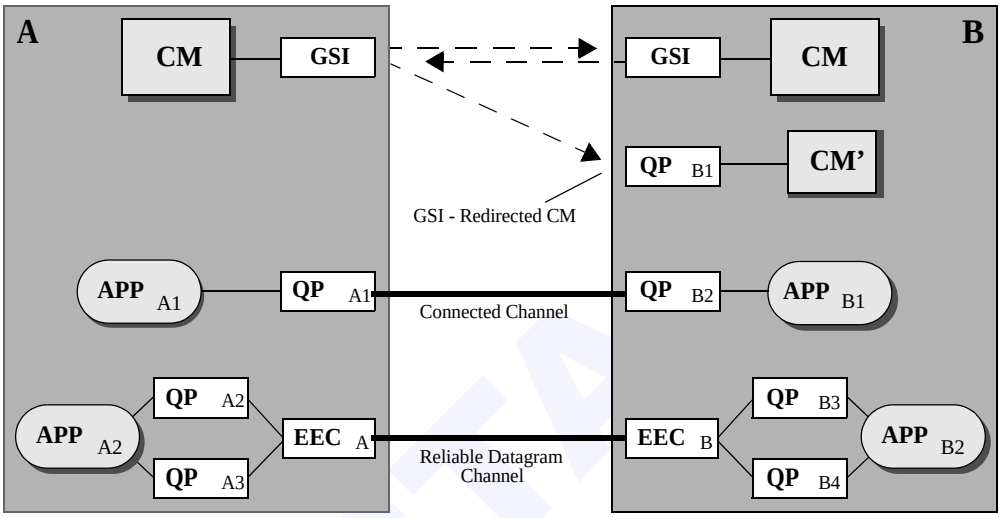


Figure 128 Communication Management Entities

Communication Management encompasses the protocols and mechanisms used to establish, maintain, and release channels for the IB Reliable Connection, Unreliable Connection, and Reliable Datagram transport service types. The Service ID Resolution Protocol (see section 12.11) enables users of Unreliable Datagram service to locate Queue Pairs supporting their desired service.

Connections are managed over Queue Pairs other than those used for the connection, through the protocol described herein, between the Communication Managers (CMs) on each system. (See Figure 128) The CMs communicate using Management Datagrams (MADs), typically over the General Services Interface (GSI) on each system. This document defines CM external behaviors, but internal interfaces and implementations are outside the scope of the InfiniBand™ Architecture specification. Examples are intended to enable understanding, not to specify implementation.

At creation, QPs and EECs are not ready for communication. The attributes of the QP/EEC must be modified (see sections 11.2.4.2 and 11.2.7.2) to support the desired communication characteristics and target(s).

Due to their nature, raw packet QPs do not need, and are not supported by, IB communication management.

The requirements on participating CMs are not equal. The initiating CM is responsible for collecting or calculating most of the information necessary to establish the connection. Much of the raw information is available from Subnet Administration, but some adjustments may be desirable, depending on the application of the channel.

CMs must maintain a certain amount of information for the lifetime of a connection. Details may be found in section 12.9.9.

CM MADs that fail the GMP check algorithm shown in Figure 169 GMP Check shall be silently dropped by the CM except where specifically noted in the sections that follow.

CM follows the standard management model documented in section 13.5.2 GSI Redirection for handling the redirection of CM requests, with the exception that the responses for CM that contain the ClassPortInfo are not of the GetResp() method, but are rather of the Send() method. (See 13.4.5 Management Class Methods for a description of the GetResp() and Send() methods. See 13.4.8.1 ClassPortInfo for a description of the redirection information inside the ClassPortInfo.) The CM messages which contain ClassPortInfo are **REJ** (see 12.6.7 REJ - Reject), **APR** (see 12.8.2 APR - Alternate Path Response) and **SIDR_REP** (see 12.11.2 SIDR_REP - Service ID Resolution Response). It should be noted that it is not an error for a requestor that has been redirected to send a subsequent request to the GSI. It is inefficient to do so, however, as there is a high likelihood that the requestor will be redirected yet again.

C12-0.1.1: All messages defined in this version of the CM Protocol shall have the ClassVersion field in the MAD header for the message set to a value of 2. All messages defined in version 1.0a of the CM Protocol that was previously published by the IBTA shall have the ClassVersion field set to a value of 1.

If a CM implementation using ClassVersion 2 sends a request (e.g. a **REQ** message) to a CM implementation that only supports ClassVersion 1, the CM running ClassVersion 1 will not be able to reply, and the request will time out. If the CM implementation using ClassVersion 2 wants to have its request successfully processed by a CM implementation using ClassVersion 1, it must send its request MAD (and other MADs) to that CM using the ClassVersion 1 format and protocol.

A CM using ClassVersion 2 can definitively make the determination that it needs to use ClassVersion 1 to correspond with a remote CM by performing a Get(ClassPortInfo) to the remote CM. The ClassVersion used in the MAD header for the Get(ClassPortInfo) MAD must be set to 1. The

ClassVersion field in the returned GetResp(ClassPortInfo) MAD can then be examined to determine if the remote CM is supporting ClassVersion 1, or both ClassVersion 1 and ClassVersion 2. See 13.4.8.1 ClassPortInfo on page 734 for details.

12.2 ESTABLISHMENT

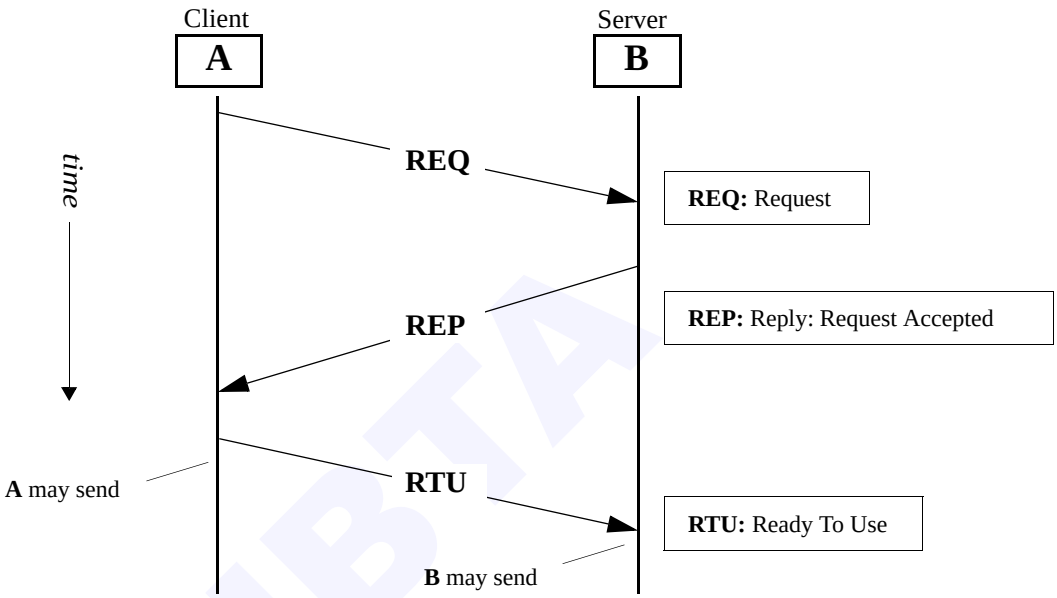


Figure 129 Sample Connection Establishment Sequence

Two models are supported by the Connection Establishment protocol: Active/Passive (Client/Server), and Active/Active (Peer to Peer).

As seen in Figure 128, the CMs on each system establish connections on behalf of their clients. The interactions between CMs and their clients are outside the scope of this specification.

In the Active/Passive model (shown in Figure 129), B's CM waits for connection requests on behalf of a server (e.g., a server process or an I/O controller) that waits (passive) for connections to be established by clients. The CM (A) for a prospective client (active) places the ServiceID that designates the desired service in the Request (**REQ**) message that begins the connection establishment sequence. The ServiceID allows the passive-side CM (B) to associate the request with the appropriate server entity. Should the **REQ** be accepted, B's CM returns the Queue Pair Number (QPN) (and End to End Context Number (EECN) for RD service) in a Response (**REP**) MAD. Whether QPs and EECs are pre-allocated or are allocated in response to a request is an implementation consideration that is outside the scope of the IBA specification.

In the Active/Active model, both entities begin as active (i.e., both send **REQ**), but one ultimately takes the passive role for establishing the connection. The selection of the passive entity is described in section 12.10.4.

Establishing a loopback connection between different QPs on the same CA is supported by the InfiniBand™ architecture. The specific mechanism used to set up a loopback connection is implementation dependent. (There are a variety of mechanisms that can be used to establish a loopback connection, including using the Connection Establishment protocol, or using the Modify Queue Pair verb to set the necessary state directly.)

12.2.1 QUIET TIME

Once a CM has initialized (which typically happens after a reboot), the CM should take care not to establish any connections until after the "quiet time". The quiet time is the interval required to remove all packets from the network that pertain to connections the CM established during its previous incarnation. If the CM does not take care to avoid establishing connections during the quiet time, it is possible that unreported data corruption may occur due to a packet from a previous connection instance being successfully received by a new connection instance.

In the worst case scenario, packets pertaining to a connection can be active in the network for a period of 2 * PortInfo:SubnetTimeout + Target ACK Delay. (See section 14.2.5.6 PortInfo for details on PortInfo:SubnetTimeout, and 12.7.33 Target ACK Delay for details on Target ACK Delay.) This is the minimum safe quiet time that the CM must wait after initialization before establishing new connections. The particular mechanism that the CM uses to wait for the quiet time is outside the scope of this specification. Possible mechanisms include:

- Having a tunable parameter that is loaded after the CM is initialized to tell the CM how long to wait.
- Having the CM store information in non-volatile storage about the maximum PortInfo:SubnetTimeout and maximum Target ACK Delay seen while the CM is running, and retrieving that information from non-volatile storage after the CM is initialized in its next incarnation.

12.3 AUTOMATIC PATH MIGRATION

The connection establishment messages specify the information necessary to support an (optional) alternate pair of endpoints to support Automatic Path Migration (APM). APM is described in section 17.2.8.1 and the support mechanisms are described in section 10.4. Channel Adapters that do not support APM may ignore the Alternate address information.

12.4 RELEASE

Connections are released through the exchange of Disconnect Request (**DREQ**) and Disconnect Reply (**DREP**) MADs. Communicating entities will likely wish to effect an orderly shutdown of their protocol before initiating the Disconnect sequence. After a connection is released, the CM shall cause each involved QP/EEC to be placed into the TimeWait state as defined in section 12.9.8.4.

CMs shall maintain enough connection state information to detect an attempt to initiate a connection on a remote QP/EEC that has not been released from a connection with a local QP/EEC, or that is in the TimeWait state. Such an event could occur if the remote CM had dropped the connection and sent **DREQ**, but the **DREQ** was not received by the local CM. If the local CM receives a **REQ** that includes a QPN (or EECN if **REQ:RDC Exists** is not set), that it believes to be connected to a local QP/EEC, the local CM shall act as defined in section 12.9.8.3.

12.4.1 STALE CONNECTION

A QP/EEC is said to have a stale connection when only one side has connection information. A stale connection may result if the remote CM had dropped the connection and sent a **DREQ** but the **DREQ** was never received by the local CM. Alternatively the remote CM may have lost all record of past connections because its node crashed and rebooted, while the local CM did not become aware of the remote node's reboot and therefore did not clean up stale connections.

12.5 SERVICE TYPES

12.5.1 SUPPORTED PROTOCOLS

The sections that follow contain message descriptions and state diagrams specifying how those messages are exchanged. The messages are used for the following purposes:

- To support connection establishment for RC and UC service types.
- To support end to end context establishment for RD service.

Figure 128 illustrates the following relationships.

12.5.2 CONNECTED SERVICES

A channel is established for Reliable Connected (RC) and Unreliable Connected (UC) service types by reaching agreement between the end CMs.

12.5.3 UNRELIABLE DATAGRAM SERVICE

Unreliable Datagram (UD) service allows a message to be sent to any destination, although there is no guarantee that the destination will re-

ceive or accept it. The ServiceID resolution facility (Section 12.11) may be used to determine the appropriate target QP.

12.5.4 RELIABLE DATAGRAM

Reliable Datagram (RD) service allows multiple Queue Pairs to communicate over a single RD channel (defined by a pair of EE contexts). One QP on each end is specified when an RD channel is established. A pair of applications using these QPs that wish to use additional QPs over that RDC do not need to use CM to associate those QPs. Application-specific messages could be sent over the original QPs to notify the other side of the QPNs of the new QPs.

Unless otherwise specified, an RD communication request implies the creation of a new RDC. Setting the **RDC Exists** field in the **REQ** message allows the sharing of the specified RDC. (See section 12.6.5) When an RDC is shared, the state of the EE contexts that comprise the channel shall not be altered, regardless of the outcome of the communication request. An RDC shall not be shared until it has been established (i.e. the CM protocol must be run to successful completion on the EE contexts comprising the RDC before an RD QP can share the RDC).

12.6 COMMUNICATION MANAGEMENT MESSAGES

The following sections describe the set of messages used to support the communication establishment scenarios supported by the IBA:

a) Active client to passive server

b) Active client to active client

c) Active client to passive server (with third-party redirector)

12.6.1 REQUIRED MESSAGES

All IBA hosts and all IBA targets that support RC, UC, or RD service types shall support the following messages:

- Request for Communication (**REQ**) (Section 12.6.5)
- Message Receipt Acknowledgement (**MRA**) (Section 12.6.6) All IBA hosts and targets are required to be able to receive and act upon an **MRA**, but the ability to send an **MRA** is optional.
- Reject (**REJ**) (Section 12.6.7)
- Reply to Request for Communication (**REP**) (Section 12.6.8)
- Ready to Use (**RTU**) (Section 12.6.9)
- Request for Communication Release (**DREQ**) (Section 12.6.10)
- Reply to Request for Communication Release (**DREP**) (Section 12.6.11)

C12-1: A CA that supports Reliable Connected, Unreliable Connected, or Reliable Datagram channels **shall** support their establishment using the CM protocol.

C12-2: For the states and messages it supports, a CM **shall** adhere to the CM protocol as defined in sections 12.9.7 and 12.9.8.

C12-3: CM message contents **shall** conform to the field descriptions in section 12.7.

C12-4: A CM **shall** support sending the **REJ** message in accordance with section 12.6.7, and **shall** support receiving the **REJ** message.

C12-5: A CM **shall**, upon receipt of an **MRA** message, behave in accordance with section 12.9.8.5.

o12-1: If a CM sends the **REQ** message, it **shall** do so in accordance with section 12.6.5.

o12-2: If a CM sends the **MRA** message, it **shall** do so in accordance with section 12.6.6.

o12-3: If a CM sends the **REP** message, it **shall** do so in accordance with section 12.6.8.

o12-4: If a CM sends the **RTU** message, it **shall** do so in accordance with section 12.6.9.

o12-5: If a CM sends the **DREQ** message, it **shall** do so in accordance with section 12.6.10.

o12-6: If a CM sends the **DREP** message, it **shall** do so in accordance with section 12.6.11.

o12-7: If a CM initiates connection requests (active role), it **shall** support sending the **REQ**, **REJ, RTU**, **DREQ**, and **DREP** messages, and responding to the **REP**, **REJ, DREQ**, and **DREP** messages.

o12-8: If a CM accepts connection requests (passive role), it **shall** support responding to the **REQ**, **REJ, RTU,** and **DREQ** messages, and sending the **REP, REJ,** and **DREP** messages.

o12-9: If a CM sends the **DREQ** message, it **shall** be able to handle the **DREP** message.

12.6.2 CONDITIONALLY REQUIRED MESSAGES

Support for these messages is required if non-management services are provided on the Channel Adapter at other than fixed QPNs. Management services include those provided through Subnet Management Packets (see 14.2 Subnet Management Class) or through General Management Packets (see Chapter 16: General Services).

- Service ID Resolution Request (**SIDR_REQ**) (Section 12.11.1)
- Service ID Resolution Response (**SIDR_REP**) (Section 12.11.2)

o12-10: If a CM sends the **SIDR_REQ** message, it **must** do so in accordance with section 12.11.1.

o12-11: If a CM sends the **SIDR_REP** message, it **must** do so in accordance with section 12.11.2.

o12-12: If a CA provides services (other than Subnet Management and General Services) using the UD service type at other than fixed QPNs, its CM **must** support receiving, processing and replying to the **SIDR_REQ** message as specified in section 12.11.

12.6.3 OPTIONAL MESSAGES

Support for these messages is optional:

- Load Alternate Path (**LAP**) (Section 12.8.1)
- Alternate Path Response (**APR**) (Section 12.8.2)

o12-13: If a CM accepts **REQ** messages and agrees to perform Automatic Path Migration, it **shall** support receiving, processing and replying to the **LAP** message as specified in section 12.8.

o12-14: If a CM sends **REQ** messages with Alternate Port/Path information, it **shall** support sending the **LAP** message as specified in section 12.8 and **shall** support receiving and processing APR messages.

12.6.4 MESSAGE USAGE

Connected Transport Service Types require state information to be established, maintained, and released at both ends of the connection. Consumers can use the messages described in this section for that purpose.

By definition, unreliable datagram communications do not require any connection state to be established, maintained, or released. However, communication services are provided to allow local and remote QPs to be associated based on a specific Service ID. (See section 12.11)

Reliable datagram communication requires Reliable Datagram Channels to be created, maintained, and released between CAs.

The Communication Management information contained in each Management Datagram message is described below. The MAD header format is defined in 16.7.1 MAD Format on page 1011.

C12-5.1.1: Except where explicitly stated otherwise in the sections that follow, the “Status” field in the MAD header shall be set to 0 for all CM MADs.

C12-5.1.2: All messages that are part of the same sequence in the CM protocol shall have the same value placed into the TransactionID field in their MAD header. The value chosen shall satisfy the uniqueness properties documented in 13.4.6.4 TransactionID usage.

There are four message sequences in the CM Protocol:

- Establishing communication: **REQ**, **REP**, **MRA** (sent either in response to **REQ** or **REP**), **REJ**, and **RTU**
- Loading an alternate path: **LAP**, **MRA** (sent in response to **LAP**), and **APR**
- Connection release: **DREQ** and **DREP**
- Service ID resolution: **SIDR_REQ** and **SIDR_REP**

C12-5.1.3: All responses generated by the CM protocol shall follow the rules for response generation that are enumerated in 13.5.4 Response Generation and Reversible Paths.

All messages in the CM protocol with the exception of the following are considered responses:

- **REQ**
- **LAP**
- **DREQ**
- **SIDR_REQ**
- **REJ** when it is sent by the active side of the protocol from the Timeout or REP Wait state as a result of a CM protocol timeout.

All of the messages in the CM protocol can contain private data. Since the CM protocol utilizes the unreliable datagram service to send its messages, it should be pointed out that successful reception of the final message in a CM protocol message exchange cannot be guaranteed. Consumers therefore cannot depend upon being able to successfully convey information in the private data of the final message in a message exchange. The final messages in the CM protocol are:

- **REJ**
- **RTU**
- **APR**
- **DREP**
- **SIDR_REP**

The messages defined below are used for both establishing connections and end to end context establishment. The message definitions are the union of the fields required for both of these purposes, and therefore there are some fields in the messages which are useful for connection establishment but not for end to end context establishment, and vice versa. This is done to decrease the total number of message types in the protocol. For each field in a message, whether the field is intended to support connection establishment or end to end context establishment (or both) is noted.

12.6.5 REQ - REQUEST FOR COMMUNICATION

REQ is sent to initiate the communication establishment sequence. The initiator (**REQ** sender) provides the Port Address (GID and/or LID) and the Queue Pair Number that it will be using for its end of the channel. For Reliable Datagram Channel establishment, the EE Context Number is included.

The initiator is responsible for proposing the Port Addresses (Primary and optional Alternate) that the target (**REQ** recipient) is to use for the channel. Based on the path defined by those port addresses, the initiator provides timeout information and the Service Level to be used by the target for any messages that it initiates. The SL from initiator to target need not be the same as from target to initiator, but the SL that is contained within the **REQ** message is the one that the initiator would prefer the target use. Path information is available from Subnet Administration (see section 15.2.5.16 PathRecord).

For service resolution and QP association over already existing Reliable Datagram Channels, **REQ:RDC Exists** must be set. When **REQ:RDC Exists** is set, the existing Reliable Datagram Channel shall not be altered either by the sender of the **REQ** or by the receiver of the **REQ**, and as a result many of the fields in the **REQ** message are not used in this case. Those fields are noted in the table below. All such unused fields shall be set to 0 by the sender, and shall be ignored by the receiver. In addition, when **REQ:RDC Exists** is set, the use of the Primary Remote/Local Port GID/LID fields is not to identify where the requestor wishes to establish the

channel--the channel is already established--but rather to identify the pair of CAs that the existing channel is operating between.

Table 99 REQ Message Contents

Field	Description	Used for Purpose	Byte [Bit] Offset	Length, bits	Values
Local Communication ID	See section 12.7.1.	C, EE	0	32	
(reserved)			4	32	
ServiceID	See section 12.7.3.	C, EE	8	64	
Local CA GUID	See section 12.7.9	C, EE	16	64	
(reserved)			24	32	
Local Q_Key	See section 12.7.13	EE	28	32	
Local QPN	See section 12.7.12.	C, EE	32	24	
Responder Resources	See section 12.7.29	C, EE	35	8	0 if RDC exists
Local EECN	See section 12.7.14	EE	36	24	
Initiator Depth	See section 12.7.30	C, EE	39	8	0 if RDC exists
Remote EECN	See section 12.7.15	EE	40	24	
Remote CM Response Timeout	See section 12.7.4	C, EE	43	5	
Transport Service Type	See section 12.7.6.	C, EE	43 [5]	2	
End-to-End Flow Control	See section 12.7.26	C	43 [7]	1	0 if RDC exists
Starting PSN	See section 12.7.31	C, EE	44	24	0 if RDC exists
Local CM Response Timeout	See section 12.7.5	C, EE	47	5	
Retry Count	See section 12.7.38	C, EE	47[5]	3	0 if RDC exists
Partition Key	See section 12.7.24	C, EE	48	16	0 if RDC exists
Path Packet Payload MTU	See section 12.7.28	C, EE	50	4	0 if RDC exists
RDC Exists	Whether RDC already exists.	EE	50[4]	1	1 if RDC exists, 0 if RDC does not
RNR Retry Count	See section 12.7.39	C, EE	50[5]	3	0 if RDC exists
Max CM Retries	See section 12.7.27	C, EE	51	4	
SRQ			51[4]	1	1 if SRQ exists 0 if SRQ does not
(reserved)			51[5]	3	
Primary Local Port LID	See section 12.7.11.	C, EE	52	16	
Primary Remote Port LID	See section 12.7.21.	C, EE	54	16	

Table 99 REQ Message Contents (Continued)

Field	Description	Used for Purpose	Byte [Bit] Offset	Length, bits	Values
Primary Local Port GID	See section 12.7.10.	C, EE	56	128	
Primary Remote Port GID	See section 12.7.20.	C, EE	72	128	
Primary Flow Label	See section 12.7.18	C, EE	88	20	0 if RDC exists
(reserved)			90[4]	6	
Primary Packet Rate	See section 12.7.25	C, EE	91[2]	6	0 if RDC exists
Primary Traffic Class	See section 12.7.17	C, EE	92	8	0 if RDC exists
Primary Hop Limit	See section 12.7.19	C, EE	93	8	0 if RDC exists
Primary SL	See section 12.7.16	C, EE	94	4	0 if RDC exists
Primary Subnet Local	See section 12.7.7	C, EE	94 [4]	1	0 if RDC exists
(reserved)			94 [5]	3	
Primary Local ACK Timeout	See section 12.7.34	C, EE	95	5	0 if RDC exists
(reserved)			95[5]	3	
Alternate Local Port LID	See section 12.7.11	C, EE	96	16	0 if RDC exists
Alternate Remote Port LID	See section 12.7.23.	C, EE	98	16	0 if RDC exists
Alternate Local Port GID	See section 12.7.10.	C, EE	100	128	0 if RDC exists
Alternate Remote Port GID	See section 12.7.22.	C, EE	116	128	0 if RDC exists
Alternate Flow Label	See section 12.7.18	C, EE	132	20	0 if RDC exists
(reserved)			134[4]	6	
Alternate Packet Rate	See section 12.7.25	C, EE	135[2]	6	0 if RDC exists
Alternate Traffic Class	See section 12.7.17	C, EE	136	8	0 if RDC exists
Alternate Hop Limit	See section 12.7.19	C, EE	137	8	0 if RDC exists
Alternate SL	See section 12.7.16	C, EE	138	4	0 if RDC exists
Alternate Subnet Local	See section 12.7.7	C, EE	138[4]	1	0 if RDC exists
(reserved)			138[5]	3	
Alternate Local ACK Timeout	See section 12.7.34	C, EE	139	5	0 if RDC exists
(reserved)			139[5]	3	
PrivateData	See section 12.7.35	C, EE	140	736	

12.6.6 MRA - MESSAGE RECEIPT ACKNOWLEDGMENT

MRA is sent in response to a **REQ** message when the recipient of the message anticipates that it will not be able to respond within the time

specified by **REQ**:Remote CM Response Timeout , or in response to a **LAP** message when the recipient of the message anticipates that it will not be able to respond within the time specified by **LAP**:Remote CM Response Timeout . It is also sent in response to a **REP** message when the recipient of the message anticipates that it will not be able to respond within the time specified by **REQ**:Local CM Response Timeout . **MRA** is sent to prevent the other party in the communication establishment protocol from either unnecessarily timing out the communication establishment attempt or flooding the link with unnecessary retries.

Table 100 MRA Message Contents

Field	Description	Used for Purpose	Byte{Bit] Offset	Length, bits	Values
Local Communication ID	See section 12.7.1.	C, EE	0	32	
Remote CommunicationID	See section 12.7.2.	C, EE	4	32	
Message MRAed	The message being MRAed.	C, EE	8	2	0x0 - **REQ**, 0x1 - **REP** 0x2 - **LAP**
(reserved)			8[2]	6	
ServiceTimeout	See section 12.7.32	C, EE	9	5	
(reserved)			9[5]	3	
PrivateData	See section 12.7.35.	C, EE	10	1776	

12.6.7 REJ - REJECT

REJ indicates that the sender will not continue through the communication establishment sequence, and the reason why it will not.

Table 101 REJ Message Contents

Field	Description	Used for Purpose	Byte[Bit] Offset	Length, bits	Values
Local Communication ID	See section 12.7.1. If this REJ is being sent with Reason code 6 (Invalid Communication ID), this field should be set to be the same as the “Remote Communication ID” field in the received message that is being rejected.	C, EE	0	32	0 if REJecting a **REQ** and no **MRA** was sent

Table 101 REJ Message Contents (Continued)

Field	Description	Used for Purpose	Byte[Bit] Offset	Length, bits	Values
Remote CommunicationID	See section 12.7.2. If this REJ is being sent with Reason code 6 (Invalid Communication ID), this field should be set to be the same as the "Local Communication ID" field in the received message that is being rejected.	C, EE	4	32	0 if REJecting due to **REP** timeout and no **MRA** was received
Message REJected	The message whose contents caused the sender to reject the communication establishment attempt. A REJ message is only sent in response to receiving a REQ, to receiving a REP, or because of a timeout occurring in the communication establishment sequence. In particular, REJ is never sent in response to receiving an MRA or RTU.	C,EE	8	2	0x0 - **REQ**. (The passive side uses this code when it receives a **REQ** and rejects the communication establishment attempt based on the contents of that **REQ**.) 0x1 - **REP**. (The active side uses this code when it receives a **REP** and rejects the communication establishment attempt based on the contents of that **REP**.) 0x2 - No message. (Both the active side and the passive side use this code when sending a **REJ** as a result of a CM protocol timeout.)
(reserved)			8[2]	6	
Reject Info Length	If non-zero, the length in bytes of valid Additional Reject Information. The sender is not required to provide Additional Reject Information even if the Reason code it places in the REJ message allows for it. If the sender decides not to provide Additional Reject Information, it shall set this field to a value of zero.	C, EE	9	7	
(reserved)			9[7]	1	

Table 101 REJ Message Contents (Continued)

Field	Description	Used for Purpose	Byte[Bit] Offset	Length, bits	Values
Reason	Error code indicating the reason for the sender's termination of the communication establishment process.	C, EE	10	16	
Additional Reject Information (ARI)	The information associated with the Reason code, as specified in section 12.6.7.2. If the number of bits of information does not completely fill the number of bytes given in Reject Info Length (e.g. if the ARI field contains a 20-bit Flow Label), the bits are packed into bytes as defined in section 1.5.1, and the remaining least significant bits of the last byte are all set to zero.	C, EE	12	576	
PrivateData	See section 12.7.35.	C, EE	84	1184	

12.6.7.1 EXAMPLE REJ MESSAGE

The content of the fields of a **REJ** that rejects a **REQ** because of an unacceptable primary port LID and suggests that a primary port LID of 200 be used are shown in the table below.

Table 102 Example REJ Message

Field	Contents
Local Communication ID	0
Remote Communication ID	0
Message REJected	0
Reason	13
Reject Info Length	2
Additional Reject Information (ARI)	200
PrivateData	empty

12.6.7.2 REJECTION REASON

Code	Reason	Description	Meaning of Additional Reject Information Field (when present)
1	No QP available	The **REQ** message required the recipient to allocate a QP, and none were available	
2	No EEC available	The **REQ** message required the recipient to allocate an EE context, and none were available	
3	No resources available	The **REQ** message required the recipient to allocate resources other than QPs or EE contexts, and none were available	
4	Timeout	The CM protocol timed out waiting for a message	Local CA GUID. The recipient of a **REJ** message with this reason code must use this CA GUID to identify the sender, as it is possible that the Remote Communication ID in the **REJ** message may not be valid.
5	Unsupported request	Receiving CM does not support this request.	
6	Invalid Communication ID	The recipient received a CM message in which the Local Communication ID, Remote Communication ID, or both, were invalid.	
7	Invalid Communication Instance	The Local Communication ID, Remote Communication ID, QPN/EECN tuple does not refer to any valid communication instance.	
8	Invalid Service ID	The recipient of the **REQ** message does not recognize or does not support the service associated with the specified ServiceID	
9	Invalid Transport Service Type	The recipient of the **REQ** message does not recognize or does not support the requested Transport Service Type	
10	Stale connection	The recipient of the **REQ/REP** determined that it already had a connection with the “Local QPN” or “Local EECN” specified in the **REQ/REP**. Upon receiving a **REJ** with this reason, the **REJ** recipient shall cause the QP or EE context to be placed into the TimeWait state as described in section 12.9.8.4.	

Code	Reason	Description	Meaning of Additional Reject Information Field (when present)
11	RDC does not exist	The Reliable Datagram Channel described in the **REQ** (Local EECN/Remote EECN) does not exist at the CA identified by the Primary Remote Port LID (or Primary Remote Port GID when this is valid) specified in the **REQ**.	
12	Primary Remote Port GID rejected	The recipient of the **REQ** message could not (or would not) accept the Primary Remote Port GID	GID of acceptable port.
13	Primary Remote Port LID rejected	The recipient of the **REQ** message could not (or would not) accept the Primary Remote Port LID	LID of acceptable port.
14	Invalid Primary SL	The recipient of the **REQ** message does not support the requested Primary SL	Acceptable SL.
15	Invalid Primary Traffic Class	The recipient of the **REQ** message does not support the requested Primary Traffic Class	Acceptable Traffic Class
16	Invalid Primary Hop Limit	The recipient of the **REQ** message could not (or would not) accept the Primary Hop Limit	Acceptable Hop Limit
17	Invalid Primary Packet Rate	The recipient of the **REQ** message could not adjust its transmitter to send as slowly as would be required to comply with the requested Primary Packet Rate	Minimum acceptable Packet Rate
18	Alternate Remote Port GID rejected	The recipient of the **REQ** message could not (or would not) accept the Alternate Remote Port GID	GID of acceptable port.
19	Alternate Remote Port LID rejected	The recipient of the **REQ** message could not (or would not) accept the Alternate Remote Port LID	LID of acceptable port.
20	Invalid Alternate SL	The recipient of the **REQ** message does not support the requested Alternate SL	Acceptable SL.
21	Invalid Alternate Traffic Class	The recipient of the **REQ** message does not support the requested Alternate Traffic Class	Acceptable Traffic Class
22	Invalid Alternate Hop Limit	The recipient of the **REQ** message could not (or would not) accept the Alternate Hop Limit	Acceptable Hop Limit
23	Invalid Alternate Packet Rate	The recipient of the **REQ** message could not adjust its transmitter to send as slowly as would be required to comply with the requested Alternate Packet Rate	Minimum acceptable Packet Rate

Code	Reason	Description	Meaning of Additional Reject Information Field (when present)
24	Port and CM Redirection	The recipient of the **REQ** message supports the requested Service ID, but at the endpoint specified by the ARI. Further CM messages should be sent to that endpoint as well.	A ClassPortInfo data structure as documented in Section 13.4.8.1 describing where to send subsequent CM messages, and also describing the GID of the port to propose in the new **REQ**.
25	Port Redirection	The recipient of the **REQ** message supports the requested Service ID, but at the port specified by the ARI. Further CM messages shall be sent to the port to which the original **REQ** was sent.	GID of port to propose in new **REQ**.
26	Invalid Path MTU	The recipient of the **REQ** message cannot support the maximum packet payload size specified	Maximum acceptable packet payload size
27	Insufficient Responder Resources	The value of Responder Resources (for RDMA Read/Atomics) in the **REP** message was insufficient.	
28	Consumer Reject	The consumer decided to reject the communication or EE context setup establishment attempt for reasons other than those listed in the other **REJ** codes. Typically this happens based upon information being conveyed in the PrivateData field of a message. It can also happen because the Consumer decided for reasons unrelated to any CM message it received to terminate the communication or EE context setup establishment attempt. This would therefore be the appropriate Reason code to use if the Consumer decided to destroy the QP or EEC in the midst of the communication or EE context setup establishment attempt.	Defined by the consumer
29	RNR Retry Count Reject	The recipient of the message rejects the RNR NAK Retry count value.	
30	Duplicate Local Communication ID	The recipient of the **REQ** message determined that it already had a connection with the sender that was using the same Local Communication ID as was specified in the **REQ**.	
31	Unsupported Class Version	The recipient of the **REQ** message does not support the Class Version specified in the header for the MAD containing the **REQ**. When returning this code, the “Code for Invalid Field” portion of the Status field in the MAD header associated with this **REJ** message should contain a value of 1. (The class version specified is not supported.) See Table 115 MAD Common Status Field Bit Values for details.	Highest CM Class Version that is supported

Code	Reason	Description	Meaning of Additional Reject Information Field (when present)
32	Invalid Primary Flow Label	The recipient of the **REQ** message does not support the requested Primary Flow Label	Acceptable Flow Label
33	Invalid Alternate Flow Label	The recipient of the **REQ** message does not support the requested Alternate Flow Label	Acceptable Flow Label

12.6.8 REP - REPLY TO REQUEST FOR COMMUNICATION

REP is returned in response to **REQ**, indicating that the respondent accepts the ServiceID, proposed primary port, and any parameters specified in the PrivateData area of the **REQ**.

When **REQ:RDC Exists** is set in the **REQ** to which the **REP** is responding, the existing Reliable Datagram Channel shall not be altered either by the sender of the **REP** or by the receiver of the **REP**, and as a result many of the fields in the **REP** message are not used in this case. Those fields are noted in the table below. All such unused fields shall be set to 0 by the sender, and shall be ignored by the receiver.

Table 103 REP Message Contents

Field	Description	Used for Purpose	Byte[Bit] Offset	Length, bits	Values
Local Communication ID	See section 12.7.1.	C, EE	0	32	
Remote Communication ID	See section 12.7.2.	C, EE	4	32	Value present in **REQ**
Local Q_Key	See section 12.7.13	EE	8	32	
Local QPN	See section 12.7.12.	C, EE	12	24	
(reserved)			15	8	
Local EE Context Number	See section 12.7.14	EE	16	24	
(reserved)			19	8	
Starting PSN	See section 12.7.31	C, EE	20	24	0 if RDC exists
(reserved)			23	8	
Responder Resources	See section 12.7.29	C, EE	24	8	0 if RDC exists
Initiator Depth	See section 12.7.30	C,EE	25	8	0 if RDC exists
Target ACK Delay	See section 12.7.33	C, EE	26	5	0 if RDC exists

Table 103 REP Message Contents (Continued)

Field	Description	Used for Purpose	Byte[Bit] Offset	Length, bits	Values
Failover Accepted	See section 12.7.36.	C, EE	26[5]	2	0: Failover accepted, or RDC exists 1: Failover port rejected because failover is not supported. Alternate Path parameters were not checked. 2: Failover is supported and all Alternate Path parameters are valid, but the failover port was rejected for some other reason.
End-To-End Flow Control	See section 12.7.26	C	26[7]	1	0 if RDC exists
RNR Retry Count	See section 12.7.39	C,EE	27	3	0 if RDC exists
SRQ			27[3]	1	1 if SRQ exists 0 if SRQ does not
(reserved)			27[2]	4	
Local CA GUID	See section 12.7.9	C, EE	28	64	
PrivateData	See section 12.7.35	C, EE	36	1568	

12.6.9 RTU - READY TO USE

RTU indicates that the connection is established, and that the recipient may begin transmitting.

Table 104 RTU Message Contents

Field	Description	Used for Purpose	Byte[Bit] Offset	Length, Bits
Local Communication ID	See section 12.7.1.	C, EE	0	32
Remote CommunicationID	See section 12.7.2.	C, EE	4	32
PrivateData	See section 12.7.35	C, EE	8	1792

12.6.10 DREQ - REQUEST FOR COMMUNICATION RELEASE (DISCONNECTION REQUEST)

DREQ is sent to initiate the connection release sequence. **DREQ** is only sent to release connections between RC QPs, UC QPs, or EE contexts.

Table 105 DREQ Message Contents

Field	Description	Used for Purpose	Byte[Bit] Offset	Length, bits
Local Communication ID	See section 12.7.1.	C, EE	0	32
Remote CommunicationID	See section 12.7.2.	C, EE	4	32
Remote QPN/EECN	See section 12.7.37	C, EE	8	24
(reserved)			11	8
PrivateData	See section 12.7.35	C, EE	12	1760

The values for Local and Remote Communication ID are those that were used to create the channel.

12.6.11 DREP - REPLY TO REQUEST FOR COMMUNICATION RELEASE

DREP is sent in response to **DREQ**, and signifies that the sender has received the **DREQ**.

Table 106 DREP Message Contents

Field	Description	Used for Purpose	Byte[Bit] Offset	Length, bits
Local Communication ID	See section 12.7.1.	C, EE	0	32
Remote CommunicationID	See section 12.7.2.	C, EE	4	32
PrivateData	See section 12.7.35	C, EE	8	1792

12.7 MESSAGE FIELD DETAILS

The following table summarizes each of the message fields, and indicates where the consumer can find the contents necessary to populate the field.

Table 107 Message Field Origins

Field	Populated From
Local Communication ID	The consumer sending the message chooses this value. See section 12.7.1
Remote CommunicationID	The consumer replying to the message chooses this value. See section 12.7.2

Table 107 Message Field Origins (Continued)

Field	Populated From
Service ID	Assuming that the consumer uses the InfiniBand™ service naming facility, this comes from the ServiceRecord, as defined in section 15.2.5.14 ServiceRecord.
Remote CM Response Timeout	The consumer should set this field to be large enough to allow enough time under normal circumstances for the recipient to be able to process the incoming message and have the response message traverse the path between source and destination. The service time at the recipient depends upon the service being requested, but the maximum time it could take to successfully traverse the path can be found in the PathRecord as defined in section 15.2.5.16 PathRecord. (How the particular path to be used is selected is a policy decision that is left up to the consumer.) The recommended upper bound on how long it should take any manager (including the CM) in the InfiniBand™ architecture to generate a response is documented in section 13.4.6.2 Timers and Timeouts.
Local CM Response Timeout	This timeout period needs to allow for the path between the source and destination to be traversed twice, and also to allow for the **REP** message to be processed. The amount of time it takes to service the **REP** message may depend upon the service that was requested, but the maximum time it could take to successfully traverse the path can be found in the PathRecord as defined in section 15.2.5.16 PathRecord. The recommended upper bound on how long it should take any manager (including the CM) in the InfiniBand™ architecture to generate a response is documented in section 13.4.6.2 Timers and Timeouts.
Transport Service Type	The consumer sets this based upon the type of service it is requesting: Reliable Connected, Unreliable Connected, or Reliable Datagram.
Subnet Local	This can be determined by comparing the PortInfo:GidPrefix fields associated with the Local Port GID and the Remote Port GID. The PortInfo record is defined in section 15.2.5.3 PortInfoRecord.
Local CA GUID	This information can be found in the NodeInfo:NodeGUID field, as defined in section 14.2.5.3 NodeInfo. (Which CA to use is a policy decision that is left up to the consumer.)
Local Port GID	This information can be found in the GidInfo record, as defined in section 15.2.5.18 GuidInfoRecord. (Which port on the CA to use and which of the available GIDs on the chosen port to use is a policy decision that is left up to the consumer.)
Local Port LID	This information can be found in the PortInfo:LID field, as defined in section 14.2.5.6 PortInfo. (Which port on the CA to use and which of the available LIDs on the chosen port to use is a policy decision that is left up to the consumer.)
Local QPN	The consumer can determine this for an HCA by querying the Queue Pair that it is offering up for connection establishment. The Query Queue Pair verb is defined in section 11.2.4.3 Query Queue Pair. (How this information is determined for a TCA is implementation-specific.)
Local Q_Key	The consumer can determine this for an HCA by querying the Queue Pair that it is offering up for connection establishment. The Query Queue Pair verb is defined in section 11.2.4.3 Query Queue Pair. (How this information is determined for a TCA is implementation-specific.)
Local EECN	The consumer can determine this for an HCA by querying the EE Context that it is offering up for communications establishment. The Query EE Context verb is defined in section 11.2.7.3 Query EE Context. (How this information is determined for a TCA is implementation-specific.)

Table 107 Message Field Origins (Continued)

Field	Populated From
Remote EECN	The data originates on the remote end of an existing connection, and is returned to the local end in a **REP** message. It is determined by the remote end in the same manner as the Local EECN.
Service Level	This information can be found in the PathRecord:SL field, as defined in section 15.2.5.16 PathRecord.
Traffic Class	This information can be found in the PathRecord:TClass field, as defined in section 15.2.5.16 PathRecord.
Flow Label	The purpose of this field is to identify a group of packets that must be delivered in order. See section 8.3 Global Route Header for a description of how this value is chosen.
Hop Limit	This information can be found in the PathRecord:HopLimit field, as defined in section 15.2.5.16 PathRecord.
Primary Remote Port GID	This information can be found in the GidInfo record associated with the remote port, as defined in section 15.2.5.18 GuidInfoRecord. The port that should be targeted based on the service being requested can be found in the ServiceRecord, as defined in section 15.2.5.14 ServiceRecord.
Primary Remote Port LID	This information can be found in the PortInfo:LID field associated with the remote port, as defined in section 14.2.5.6 PortInfo. The port that should be targeted based on the service being requested can be found in the ServiceRecord, as defined in section 15.2.5.14 ServiceRecord.
Alternate Remote Port GID	This information can be found in the GidInfo record associated with the remote port, as defined in section 15.2.5.18 GuidInfoRecord. The port that should be targeted based on the service being requested can be found in the ServiceRecord, as defined in section 15.2.5.14 ServiceRecord.
Alternate Remote Port LID	This information can be found in the PortInfo:LID field associated with the remote port, as defined in section 14.2.5.6 PortInfo. The port that should be targeted based on the service being requested can be found in the ServiceRecord, as defined in section 15.2.5.14 ServiceRecord.
Partition Key	This information can be found in the PathRecord:P_Key field, as defined in section 15.2.5.16 PathRecord.
Packet Rate	This information can be found in the PathRecord:Rate field, as defined in section 15.2.5.16 PathRecord.
End-to-End Flow Control	This field is used for RC Service Connections. For HCAs, if the RC QP is not associated with an SRQ, the End-to-End Flow Control bit must be set to one. For HCAs, if the RC QP is associated with an SRQ, the End-to-End Flow Control bit must be set to zero. Whether or not End-to-End Flow Control is supported by a TCA is an implementation option, and it is therefore outside the scope of the InfiniBand™ architecture to specify the origin of this field in a TCA. Note: 1 means e2e flow control supported, 0 means not supported.
SRQ	This field is used for RC Service Connections. If QP is associated with an SRQ, this bit must be set to one. If the QP, is not associated with an SRQ, this bit must be set to zero.
Max CM Retries	The value of this field is a policy decision that is outside the scope of Communication Management to define. The field is discussed in section 12.7.27.

Table 107 Message Field Origins (Continued)

Field	Populated From
Path Packet Payload MTU	This information can be found in the PathRecord:Mtu field, as defined in section 15.2.5.16 PathRecord.
Responder Resources	The consumer can determine the maximum supported value for a QP/EEC by querying the HCA that will be used for communication. The Query HCA verb is defined in section 11.2.1.2 Query HCA
Initiator Depth	The consumer can determine the maximum supported value for a QP/EEC by querying the HCA that will be used for communication. The Query HCA verb is defined in section 11.2.1.2 Query HCA
Starting PSN	The value of this field is a policy decision that is outside the scope of Communication Management to define. The field is discussed in section 12.7.31.
Service Timeout	The consumer should set this field to be large enough to allow enough time for it to complete the processing of the incoming message and have the response message that it sends out traverse the path between source and destination. The incoming message processing time depends upon the service being requested and potentially other state, but the maximum time it could take to successfully traverse the path can be found in the PathRecord as defined in section 15.2.5.16 PathRecord.
Target ACK Delay	The value of this field is a policy decision that is outside the scope of Communication Management to define. The field is discussed in section 12.7.33 Target ACK Delay.
Local ACK Timeout	The value of this field is a policy decision that is outside the scope of Communication Management to define. The field is discussed in section 12.7.34 Local ACK Timeout.
PrivateData	The contents of this field are outside the scope of what the InfiniBand™ specification defines; the usage (if any) of this field is specified by higher-level communications establishment protocols.
Failover Accepted	Set as per the description in section 12.7.36 Failover Accepted.
Remote QPN/EECN	This should be the same as the Local QPN/Local EECN returned in the **REP** message.
Retry Count	The value of this field is a policy decision that is outside the scope of Communication Management to define. The field is discussed in section 12.7.38 Retry Count.
RNR Retry Count	The value of this field is a policy decision that is outside the scope of Communication Management to define. The field is discussed in section 12.7.39 RNR Retry Count.

12.7.1 LOCAL COMMUNICATION ID

An identifier that uniquely identifies this connection from the sender's point of view. The sender must use the same identifier for all phases of communication establishment and release. It must not reuse a Local Communication ID for the life of the connection, or while any messages related to the connection could still be in the fabric. (How long a message related to the connection could still be in the fabric is touched upon in section 12.9.8.4.) The Communication ID allows the recipient to determine whether the message is a duplicate of an old message, or represents a new connection request.

12.7.2 REMOTE COMMUNICATION ID

An identifier that uniquely identifies this connection from the recipient's point of view. (As an example, for a **REP** message this would be the same as the Local Communication ID that was received in the **REQ** message.) The values in the Local and Remote Communication ID fields in the Communication Management MADs are exchanged between requests and replies.

The pair of (Local Communication ID, Remote Communication ID) is used to reference connections during establishment, failover management, and release. CM messages with invalid Communication IDs shall not be processed, and shall be rejected as specified in section 12.6.7.

12.7.3 SERVICEID

An identifier that specifies the service being requested. The ServiceID field specifies the service number desired by the requestor. These include, but are not limited to, the service numbers defined for typical TCP services. The mappings between services and ServiceIDs are outside the scope of Communication Management.

12.7.4 REMOTE CM RESPONSE TIMEOUT

The time, expressed as ($4.096\ \mu S * 2^{\text{Remote CM Response Timeout}}$), within which the CM message recipient shall transmit a response to the sender. This value is unsigned. The recipient uses this information to determine whether it should send an **MRA**. (See section 12.9.8.5)

12.7.5 LOCAL CM RESPONSE TIMEOUT

The time, expressed as ($4.096\ \mu S * 2^{\text{Local CM Response Timeout}}$), that the remote CM shall wait for a response from the local CM to a CM message sent by the remote CM. This value is unsigned. Note that whereas Remote CM Response Timeout is the time between receipt of a message and transmission of a response, Local CM Response Timeout includes that “turn-around” time, as well as round trip packet flight time. (See section 12.9.8.5) The initiating CM is responsible for determining this value, through Subnet Management or other means.

12.7.6 TRANSPORT SERVICE TYPE

Specifies desired service type: Reliable Connected, Unreliable Connected, or Reliable Datagram. The field is encoded as follows:

0: RC

1: UC

2: RD

3: Reserved

12.7.7 SUBNET LOCAL

0: Local and remote CA ports are on different subnets
1: Local and remote CA ports are on the same subnet

12.7.8 THIS SECTION HAS BEEN DELETED

This section has been deleted.

12.7.9 LOCAL CA GUID

The EUI-64 GUID of the sending Channel Adapter.

12.7.10 LOCAL PORT GID

The GID of the local CA port on which the channel is to be established. This field shall be valid regardless of whether the local and remote ports are on the same subnet or different subnets. If an alternate path is not to be specified, the **Alternate Local Port GID** field shall be set to zero. If this field is non-zero, it shall contain a valid GID.

12.7.11 LOCAL PORT LID

The LID of the local CA port on which the channel is to be established. When local and remote ports are on different subnets, this field must be the LID of the router that the passive side will target for the return path. If an alternate path is not to be specified, the **Alternate Local Port LID** field shall be set to zero.

12.7.12 LOCAL QPN

The QPN of the message sender's QP on which the channel is to be established. One Reliable Datagram QP may be associated with multiple EE contexts. A QPN must be specified when establishing an RD channel, but use of this QPN is not limited to this RDC. Once a consumer establishes a Reliable Datagram Channel, the consumer may use additional QPs over the RDC without an additional connection establishment exchange.

CM shall not be used to connect the Send Work Queue of a QP to the Receive Work Queue of the same QP. (If so desired, the consumer can do this using the Modify QP verb.) Attempting to do this may result in unpredictable behavior when doing connection establishment between peers.

12.7.13 LOCAL Q_KEY

(RD Only) The Q_Key for the QP specified by **Local QPN**.

12.7.14 LOCAL EECN

The EE Context Number for the message sender's end of the RD channel.

12.7.15 REMOTE EECN

The EE Context Number for the remote end of the existing Reliable Datagram channel. 0 if **REQ:RDC Exists** is not set.

12.7.16 SERVICE LEVEL

The value to be placed in the Service Level field for packets sent by the recipient. This Service Level is the one that the initiator would prefer the target use. For more information on Service Levels, see section 7.6.5 Service Level on page 185.

12.7.17 TRAFFIC CLASS

Defines Traffic Class for globally-routed packets.

12.7.18 FLOW LABEL

Defines Flow Label for globally-routed packets.

12.7.19 HOP LIMIT

The maximum number of hops a packet can make between subnets before being discarded.

12.7.20 PRIMARY REMOTE PORT GID

The GID of the remote node's CA port on which the local node wishes to establish the channel. This field shall be valid regardless of whether the local and remote ports are on the same or different subnets. The remote node may send **REJ** to reject this port, and may optionally suggest an acceptable port.

12.7.21 PRIMARY REMOTE PORT LID

The LID of the remote node's CA port on which the local node wishes to establish the channel. The remote node may send **REJ** to reject this port, and may optionally suggest an acceptable port. The sender is responsible for ensuring that the LID and GID refer to the same port.

12.7.22 ALTERNATE REMOTE PORT GID

As in section 12.7.20. A CA that does not support automatic failover shall set the **REP**'Failover Accepted' field to one. If this field is non-zero, it shall contain a valid GID.

12.7.23 ALTERNATE REMOTE PORT LID

As in section 12.7.21. A CA that does not support automatic failover shall set the **REP** 'Failover Accepted' field to one.

12.7.24 PARTITION KEY

The Partition Key to be used for the channel being established.

12.7.25 PACKET RATE

The maximum rate for the proposed path as obtained from the PathRecord:Rate field as defined in section 15.2.5.16 PathRecord.

12.7.26 END-TO-END FLOW CONTROL

Signifies whether the local CA actually implements End-to-End Flow Control (1), or instead always advertises 'invalid credits'(0). See section 9.7.7.2 End-to-End (Message Level) Flow Control for more detail.

12.7.27 MAX CM RETRIES

Maximum number of times that either party can re-send a **REQ**, **REP**, or **DREQ** message. After re-sending for the maximum number of times without a response, the sending party should then terminate the protocol by sending a **REJ** message indicating that it timed out.

12.7.28 PATH PACKET PAYLOAD MTU

Specifies the maximum packet payload size, in bytes, for the channel being established. One of 256, 512, 1024, 2048, 4096. This value applies to both the primary and alternate paths.

12.7.29 RESPONDER RESOURCES

The maximum number of outstanding RDMA Read/Atomic operations the sender will support from the remote QP/EEC. This value may be zero. The maximum number that the HCA can support for a QP/EEC can be determined using the Query HCA verb. See section 11.2.1.2 Query HCA.

The recipient of the **REQ** message shall choose a local Initiator Depth that does not exceed the Responder Resources offered in the **REQ**. If the recipient of the **REQ** message is unwilling or unable to do so, it shall send a **REJ** message to discontinue the connection establishment.

The recipient of the **REP** message shall decide whether the Responder Resources offered in the **REP** are sufficient for the Initiator Depth the recipient of the **REP** wishes to use. If not, it shall send a **REJ** message to discontinue the connection establishment.

12.7.30 INITIATOR DEPTH

The maximum number of outstanding RDMA Read/Atomic operations the sender will have to the remote QP/EEC. This value may be zero. The maximum number that the HCA can support for a QP/EEC can be determined using the Query HCA verb. See section 11.2.1.2 Query HCA.

The recipient of the **REQ** message should try to choose a number of local Responder Resources that is greater than or equal to the Initiator Depth in the **REQ** message. If it is unwilling or unable to do so, it may send a

REP message containing fewer Responder Resources than the Initiator Depth in the **REQ** message.

12.7.31 STARTING PSN

The transport Packet Sequence Number at which the remote node (relative to the sender of the **REQ** or **REP** message) shall begin transmitting over the newly established channel. This value should be chosen to minimize the chance that a packet from a previous connection could fall within the valid PSN window.

12.7.32 SERVICE TIMEOUT

Present in the **MRA**. The maximum time required for the sender to send a **REP**, **RTU**, **APR**, or **REJ** (as appropriate). This value is expressed as ($4.096\ \mu S * 2^{\text{Service Timeout}}$) from the time the **MRA** is posted to the Send queue. The recipient of the MRA shall wait the specified time, plus a packet lifetime, after receiving this message before timing out. (See section 12.9.8.5) This value is unsigned.

12.7.33 TARGET ACK DELAY

($4.096\ \mu S * 2^{\text{Target ACK Delay}}$) represents the maximum expected time interval between the target CA's reception of a message and the transmission of the associated ACK or NAK. This is information furnished by the target to the recipient. It provides the recipient with information about the maximum message processing latency of the target, which is one component of the overall time it takes to get an ACK or NAK after having sent a request packet. (The other component is the network propagation delay, which depends upon the configuration of the switches and routers between the two endpoints as well as the congestion in the network.) The recipient of the message containing the Target ACK Delay should use this value along with the recipient's best estimate of the network propagation delay to determine how long to wait before timing out a packet transmission to the target. This value is unsigned.

12.7.34 LOCAL ACK TIMEOUT

Value representing the transport (ACK) timeout for use by the remote, expressed as ($4.096\ \mu S * 2^{\text{Local ACK Timeout}}$). Calculated by **REQ** sender, based on (2 * PacketLifeTime + Local CA's ACK delay). Although the remote CA is not required to use this value for its ACK timeout, it is strongly encouraged to do so. PacketLifeTime represents the maximum expected time interval consumed by a packet traversing the path between source and destination CA. PacketLifeTime is contained in the PathRecord, as defined in section 15.2.5.16 PathRecord. Local ACK Timeout is unsigned.

If too small a value is chosen for the Local ACK Timeout, the number of packet transmission timeouts reported by the remote CA may increase,

which may increase the amount of work that is required in the CA to successfully send a packet. If too large a value is chosen, the amount of time that it takes to notice that a packet has not been successfully transmitted (e.g. due to a CRC error on the wire) will be increased, which may increase the amount of time it takes to recover from or report such errors.

12.7.35 PRIVATEDATA

Data that is opaque to the communication management protocol, passed from the sender to the recipient. The recipient may choose to accept or reject the request based on the private data. The format and meaning of the PrivateData field is specific to the ServiceID and message type, and is not specified within Communication Management.

12.7.36 FAILOVER ACCEPTED

Indicates whether the target of the **REQ** accepted or rejected the Alternate port address contained in the **REQ**. By sending the **REP**, the target accepts the connection request, but it may still reject the proposed failover port.

If failover is accepted, each CM shall cause the associated QP (for RC/UC) or EEC (for RD) specified by Local QPN to be placed in the REARM Migration State (see section 17.2.8.1 Automatic Path Migration Protocol).

If failover is rejected, each CM shall cause the associated QP or EEC to be placed in the Migstate:MIGRATED state upon transition to the RTR state.

12.7.37 REMOTE QPN/EECN

The remote (relative to the sender) QPN or EECN, as appropriate, that is the subject of the message. Provides an additional check that the (Local Communication ID, Remote Communication ID) pair references the correct resource.

12.7.38 RETRY COUNT

The total number of times that the sender wishes the receiver to retry timeout, packet sequence, etc. errors before posting a completion error. See sections 9.9.2.1.1 Requester Error Retry Counters and 9.9.2.4.1 Requester Class A Fault Behavior for details of how the retry counter works.

12.7.39 RNR RETRY COUNT

The total number of times that the **REQ** or **REP** sender wishes the receiver to retry RNR NAK errors before posting a completion error. See sections 9.9.2.1.1 Requester Error Retry Counters and 9.9.2.4.1 Requester Class A Fault Behavior for details of how the RNR retry counter works.

12.8 ALTERNATE PATH MANAGEMENT

IBA supports Automatic Path Migration (see section 17.2.8 Automatic Path Migration), in which a channel's traffic (RC, UC, RD) may be moved to a pre-determined alternate path. The initial alternate path is established at connection setup, but if a migration occurs, a new path needs to be specified before re-enabling migration.

Two messages are specific to alternate path management. **LAP - Load Alternate Path** carries the new path information. **APR - Alternate Path Response** informs the requester of the status of the **LAP** request. The requestor (i.e. the sender of the LAP message) must be the same side of the channel that was in the active/client role when the CM state for the channel changed to Established; the passive side of the channel shall not send a LAP message.

The **MRA** message may be sent by the **LAP** recipient if it is unable to send the **APR** message within the Remote CM Response Timeout . As the **LAP** is idempotent, the message may re-sent if there is no response, or if the Service Timeout is not met. The recipient **shall** return a failure status in the **APR** if the **LAP** request specifies an alternate path that is the same, in every respect, as the primary path. There is no limit on the number of **LAP** messages that a sender may have outstanding, but a sender shall have no more than one **LAP** outstanding per remote QP/EEC at any time.

The QP/EEC state changes requested by the **LAP** and **APR** messages may be effected through the ModifyQP or ModifyEE verbs (sections 11.2.4.2 and 11.2.7.2). Only RC QP, UC QP, or EE context state shall be modified; the **LAP** and **APR** messages do not modify the state associated with RD QPs.

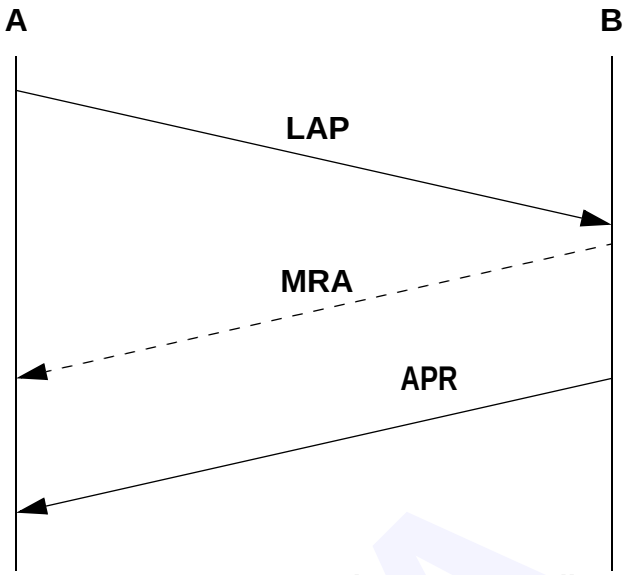


Figure 130 Loading alternate path

12.8.1 LAP - LOAD ALTERNATE PATH

LAP is an optional message used to change the alternate path information for a specific connection. It may be sent to update the alternate path information if fabric changes cause it to become invalid, or to load the "new" alternate path information after a path migration occurs. Loading alternate path information does not initiate the migration process for automatic failover; it just specifies which path is to be used when the path migration occurs.

Table 108 LAP Message Contents

Field	Description	Byte [Bit] Offset	Length, bits
Local Communication ID	See section 12.7.1.	0	32
Remote Communication ID	See section 12.7.2.	4	32
(reserved)		8	32
Remote QPN/EECN	See section 12.7.37	12	24
Remote CM Response Timeout	See section 12.7.4	15	5
(reserved)		15[5]	3
(reserved)		16	32
Alternate Local Port LID	See section 12.7.11	20	16
Alternate Remote Port LID	See section 12.7.23.	22	16

Table 108 LAP Message Contents (Continued)

Field	Description	Byte [Bit] Offset	Length, bits
Alternate Local Port GID	See section 12.7.10.	24	128
Alternate Remote Port GID	See section 12.7.22.	40	128
Alternate Flow Label	See section 12.7.18	56	20
(reserved)		58[4]	4
Alternate Traffic Class	See section 12.7.17	59	8
Alternate Hop Limit	See section 12.7.19	60	8
(reserved)		61	2
Alternate Packet Rate	See section 12.7.25	61[2]	6
Alternate SL	See section 12.7.16	62	4
Alternate Subnet Local	See section 12.7.7	62[4]	1
(reserved)		62[5]	3
Alternate Local ACK Timeout	See section 12.7.34	63	5
(reserved)		63[5]	3
Private Data	See section 12.7.35	64	1344

12.8.2 APR - ALTERNATE PATH RESPONSE

APR is sent in response to a **LAP** request. **MRA** may be sent to allow processing of the **LAP**.

Table 109 APR Message Contents

Field	Description	Byte[Bit] Offset	Length, bits
Local Communication ID	See section 12.7.1.	0	32
Remote Communication ID	See section 12.7.2.	4	32
Additional Information Length	If non-zero, the length in bytes of valid Additional Information. The sender is not required to provide Additional Information even if the AP status code it places in the APR message allows for it. If the sender decides not to provide Additional Information, it shall set this field to a value of zero.	8	8

Table 109 APR Message Contents (Continued)

Field	Description	Byte[Bit] Offset	Length, bits
AP status	See section 12.8.2.1	9	8
(reserved)		10	16
Additional Information	The information associated with the AP status code, as specified in section 12.8.2.1. If the number of bits of information does not completely fill the number of bytes given in Additional Information Length (e.g. if the Additional Information field contains a 20-bit Flow Label), the bits are packed into bytes as defined in section 1.5.1, and the remaining least significant bits of the last byte are all set to zero.	12	576
Private Data	See section 12.7.35	84	1184

12.8.2.1 AP STATUS

Value	Description	Meaning of Additional Information Field (when present)
0	Alternate path information loaded	
1	Invalid Remote/Local Communication ID	
2	Alternate paths not supported. Alternate path parameters were not checked.	
3	Alternate paths are supported and all Alternate Path parameters are valid, but the failover port was rejected for some other reason.	
4	Alternate path information rejected - redirect	A ClassPortInfo data structure as described in section 13.4.8.1 ClassPortInfo on page 734. The **LAP** sender may use the information in the ClassPortInfo to resend the **LAP** to an entity that can process that message.
5	Proposed alternate path matches current primary path	
6	Remote QPN/EECN does not match with connection identified by Local/Remote Communication ID	
7	Proposed Alternate Remote Port LID rejected	LID of acceptable port

Value	Description	Meaning of Additional Information Field (when present)
8	Proposed Alternate Remote Port GID rejected	GID of acceptable port
9	Proposed Alternate Flow Label rejected	Acceptable Flow Label
10	Proposed Alternate Traffic Class rejected	Acceptable Traffic Class
11	Proposed Alternate Hop Limit rejected	Acceptable Hop Limit
12	Proposed Alternate Packet Rate rejected	Minimum acceptable Packet Rate
13	Proposed Alternate SL rejected	Acceptable SL

12.9 STATE TRANSITION DIAGRAMS FOR COMMUNICATION ESTABLISHMENT AND RELEASE

The diagrams in this section detail all valid states and state transitions in the IBA communication establishment and release protocols. Section 12.10 contains ladder diagrams which illustrate various paths through this state diagram.

The InfiniBand™ communication establishment and communication release protocols are structured so that they will always run to completion in a bounded amount of time. “Completion” for the communication establishment protocol means that the communication will either be established, or else the state of all parties involved in the communication will revert to idle as if no communication had ever been established. “Completion” for the communication release protocol means that the communication is released; this protocol never fails to run to completion.

The diagrams in this section assume that the Consumer runs the IBA communication establishment and release protocols to completion (successful or otherwise). It is legal, however, for the Consumer to destroy the QP or EEC involved in the communication establishment/release attempt before that attempt has run to completion. If this occurs, the Consumer whose QP or EEC has been destroyed has one of two options: they can ignore all incoming messages that pertain to the communication establishment/release attempt involving the QP or EEC (and let the communications establishment attempt timeout), or they can send a **REJ** message with a “Consumer Reject” Reason code if they are in a CM state (i.e. REP Rcvd, MRA(REP) Sent, REQ Rcvd, MRA Sent) that allows a **REJ** to be sent.

12.9.1 DIAGRAM DESCRIPTION

There is only one communication establishment protocol for InfiniBand™, with different messages used for different scenarios. The state diagrams

are broken into an “active side” and a “passive side”. The active side of the protocol is the side that is trying to initiate a transition out of one of the terminal states (Idle and Established). The passive side of the protocol is the side that is responding to the active side.

12.9.2 INVALID STATE INPUT HANDLING

Several of the states in the CM protocols are ephemeral; the transition out of these states depends only upon a decision being made locally, not on any new input being received from the other party in the communication establishment or release attempt. The ephemeral states are:

- Peer Compare
- Timeout
- REJ Retry
- REP Rcvd
- DREQ Rcvd
- DREP Timeout
- REQ Rcvd
- REJ Sent
- RTU Timeout

The general rule for handling any input message that is received while in an ephemeral state is to hold that message pending until the CM protocol enters a non-ephemeral state.

In many of the non-ephemeral states of the InfiniBand™ communication establishment and release protocols, there is a defined set of input messages that can legally be received while in that state. Once the CM protocol has entered a non-ephemeral state, the general rule for handling an input message that cannot be legally received and acted upon while in that state is to ignore it. A CM shall not retry the **REQ**, **REP**, or **DREQ** messages more than the number of times specified by **REQ:Max CM Retries**.

12.9.3 TIMEOUTS

A lost or dropped message will ultimately result in a timeout. Since all parties will ultimately return to the idle state, there is no correctness requirement to do retries of a message send as a result of a timeout, although it is recommended. Senders of retried messages may not modify the contents of the messages between retries.

In the following state diagrams, "Timeout" represents a Response Timeout. Service Timeouts are specifically noted.

12.9.4 STATE DIAGRAM NOTES

All **REJ**s, sent or received, cause a return to IDLE(active) or LISTEN(passive), possibly through the TimeWait state (see section 12.9.8.4).

In the Active Communication Establishment diagram, the transition from the Peer Compare state to the Passive REQ_Rcvd state only happens if the ServiceID in the **REQ** received is the same as the ServiceID in the **REQ** that was sent. (See section 12.10.4 for details). Otherwise, a new connection establishment instance shall be started.

The ServiceID implicitly defines whether the service is client/server or peer to peer, but the server application must inform its CM so that the CM will handle the inbound **REQ** correctly.

When the Consumer wishes to destroy a QP or EEC that is in the Established CM state, it is good practice for the Consumer to first release the connection before destroying the QP or EEC. Doing so allows any state maintained by CM related to the QP or EEC in question to be cleaned up. A connection is released by moving from the Established state to the TimeWait state using one of the state transition sequences described in the sections that follow.

When the CM state is IDLE, LISTEN, or TimeWait, the QP or EE Context is allowed to be in any of the Error, Reset, or Initialized states. In these CM states, which particular state the QP or EE Context is in is outside the scope of this specification. The CM should avoid transitioning a QP from the Error state to the Reset state before it has notified the Consumer that the QP has entered the Error state, and given the Consumer an opportunity to dequeue any Work Completions that may be associated with that QP. If the CM doesn't follow this suggestion the Consumer may be unable to dequeue the Work Completions associated with a QP after the QP is transitioned to the Reset state.

12.9.5 COMMUNICATION ESTABLISHMENT AND RELEASE - ACTIVE

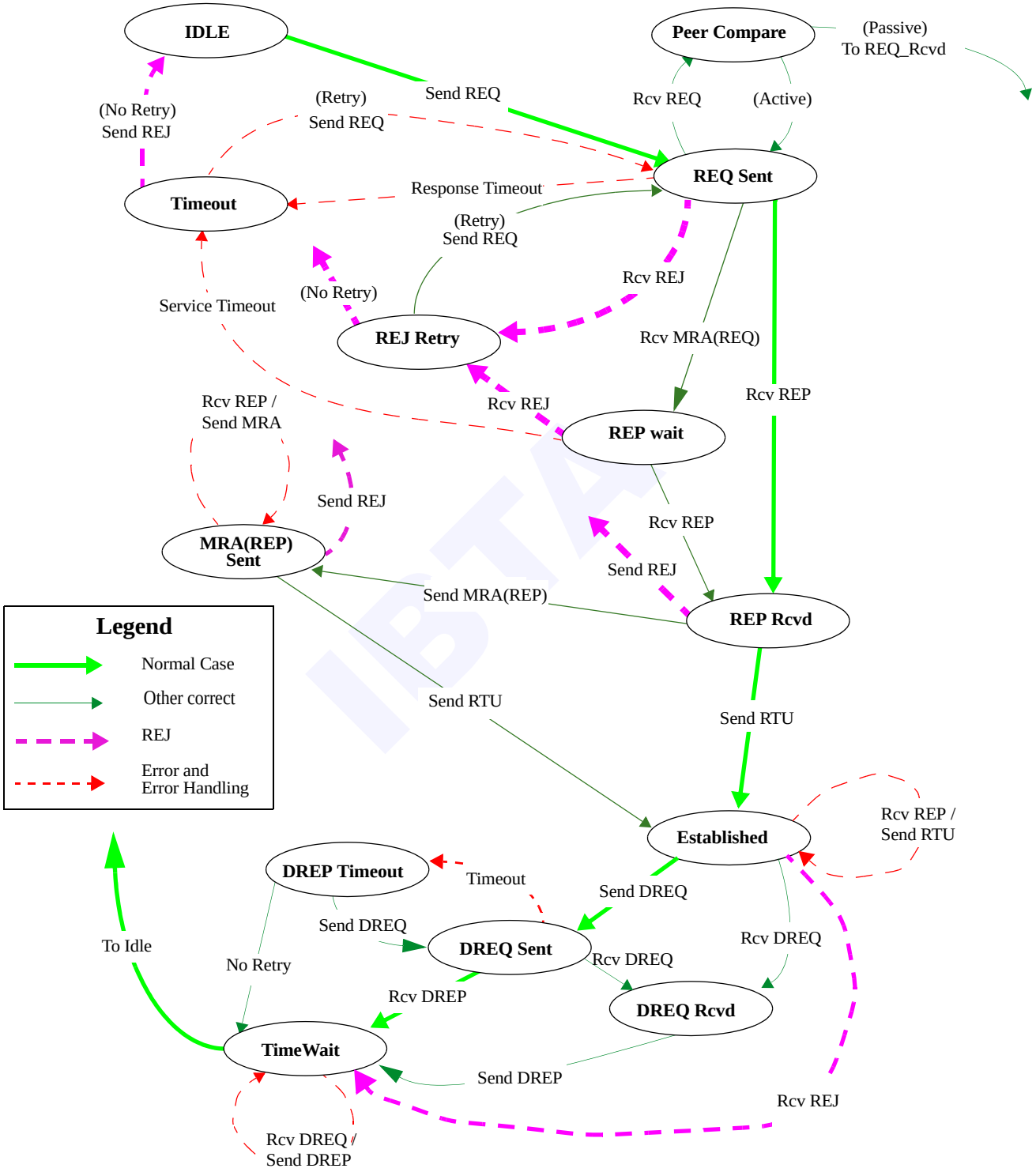


Figure 131 Communication Establishment(Active Side)

12.9.6 COMMUNICATION ESTABLISHMENT - PASSIVE

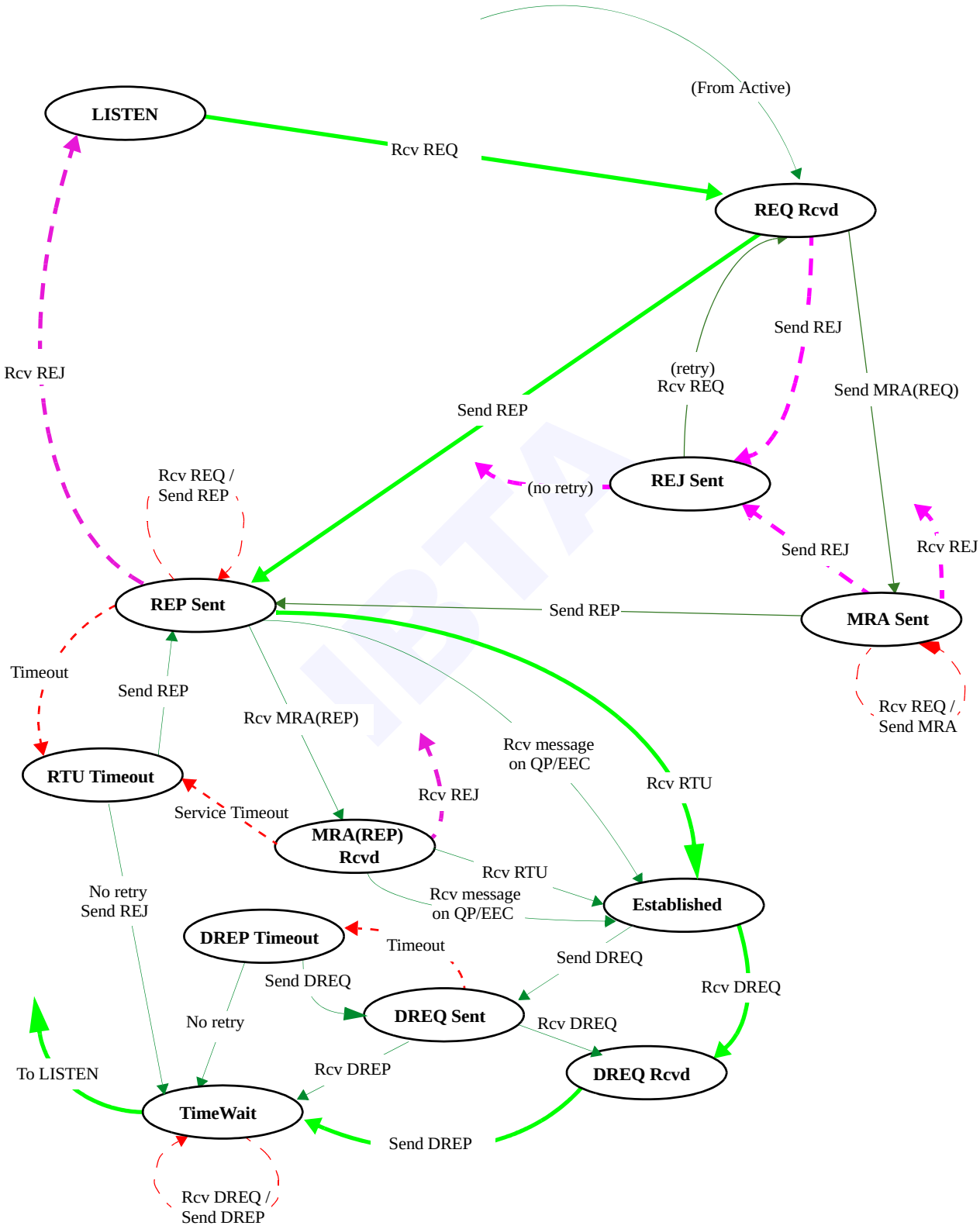


Figure 132 Communication Establishment(Passive Side)

12.9.7 STATE AND TRANSITION DEFINITIONS

The following tables define each state and the possible transitions from the state.

These tables define the protocol, and take precedence in the case of a conflict with the state diagrams.

In this table, “CEP” (Channel EndPoint) means QP when the protocol is being used to establish or release connections between RC or UC QPs. It means EE context when the protocol is being used to establish an RDC between two EE contexts. If the protocol is being used to advertise an RD QP for use over an existing RDC (i.e. if **REQ:RDC Exists** is set), there is no CEP, and hence the state transitions to the CEP called out in this table do not apply in this case. (The CM protocol state transitions continue to apply, however.)

12.9.7.1 ACTIVE STATES

CM State	Event	Action/Transition Sequence
IDLE		
	Send REQ	Send REQ / CM to REQ Sent / *CEP to Initialized*
	(default)	None
REQ Sent		
	Receive REP	CM to REP Rcvd / *CEP to Ready to Receive*
	Receive REQ	IF (ServiceIDs match) to Peer Compare
	Receive MRA(REQ)	CM to REP wait
	Response Timeout	CM to Timeout
	Receive REJ	CM To REJ Retry
	(default)	None
Peer Compare		

	Entry	IF ((local CA GUID greater than remote CA GUID) OR
		((CEP is a QP) AND (local QPN greater than remote QPN)) OR
		((CEP is an EEC) AND (local EECN greater than remote EECN)))
		CM to REQ Sent
		ELSE
		CM to Passive:REQ Rcvd

REP wait		

CM State	Event	Action/Transition Sequence
	Receive REP	CM to REP Rcvd / *CEP to Ready to Receive*
	Service Timeout	CM to Timeout
	Receive REJ	CM to REJ Retry
	(default)	None
REJ Retry		
	Entry (Retry)	Send REQ / CM to REQ Sent
	Entry (No Retry)	CM to IDLE / *CEP to Error*
REP Rcvd		
	Send RTU	Send RTU / CM to Established / *CEP to Ready to Send*
	Send MRA(REP)	CM to MRA(REP) Sent
	Send REJ	CM to IDLE / *CEP to Error*
	(default)	None
MRA(Rep) sent		
	Send RTU	Send RTU / CM to Established / *CEP to Ready to Send*
	Send REJ	CM to IDLE / *CEP to Error*
	Receive REJ	CM to IDLE / *CEP to Error*
	Receive REP	Send MRA
	(default)	None
Established		
	Receive DREQ	CM to DREQ Rcvd / *CEP to Error*
	Send DREQ	CM to DREQ Sent / *CEP to Error*
	Receive REQ	See section 12.9.8.3.1
	Receive REP	Send RTU
	Receive REJ	CM to TimeWait / *CEP to Error*
	(default)	None
DREQ Sent		
	Timeout	CM to DREP Timeout
	Receive DREQ	CM to DREQ Rcvd

CM State	Event	Action/Transition Sequence
	Receive DREP	CM to TimeWait
	(default)	None
DREQ Rcvd		
	Send DREP	CM to TimeWait
	(default)	None
TimeWait		
	Entry	CM: Start Timer
	Receive DREQ	CM: Send DREP (if Max CM Retries not exceeded)
	Timer Expiration	CM to IDLE
	(default)	None
Timeout		
	Entry (Retry) (Max Retries not exceeded)	Send REQ / CM to REQ Sent
	Entry (No Retry)	Send REJ / CM to IDLE / *CEP to Error*
DREP Timeout		
	Entry (Retry) (Max Retries not exceeded)	Send DREQ / CM to DREQ Sent
	Entry (No Retry)	CM to TimeWait
	(default)	None

12.9.7.2 PASSIVE STATES

State	Event	Action/Transition Sequence
LISTEN		
	Receive REQ	*CEP to Initialized* / CM to REQ Rcvd
	(default)	None
REQ Rcvd		
	Send REP	*CEP to Ready to Receive* / Send REP / CM to REP Sent
	Send MRA(REQ)	CM to MRA Sent

State	Event	Action/Transition Sequence
	Send REJ	CM to REJ Sent
	(default)	None
MRA sent		
	Send REP	*CEP to Ready to Receive* / Send REP / CM to REP Sent
	Send REJ	CM to REJ Sent
	Receive REJ	CM to LISTEN / *CEP to Error*
	Receive REQ	Send MRA
	(default)	None
REJ Sent		
	(retry) Receive REQ	CM to REQ Rcvd
	(no retry)	CM to LISTEN / *CEP to Error*
REP Sent		
	Receive RTU	CM to Established / *CEP to Ready to Send*
	Receive MRA(REP)	CM to MRA(REP) Rcvd
	Receive message on service CEP	CM To Established / *CEP to Ready to Send*
	Receive REJ	CM to Listen / *CEP to Error*
	Timeout	CM to RTU Timeout
	Receive REQ	Send REP
	(default)	None
MRA(Rep) rcvd		
	Receive RTU	CM to Established / *CEP to Ready to Send*
	Service Timeout	CM to RTU Timeout
	Receive message on service CEP	CM To Established / *CEP to Ready to Send*
	Receive REJ	CM to LISTEN / *CEP to Error*
	(default)	None
Established		
	Send DREQ	CM to DREQ Sent / *CEP to Error*
	Receive DREQ	CM to DREQ Rcvd / *CEP to Error*

State	Event	Action/Transition Sequence
	Receive REQ	See section 12.9.8.3.1
	(default)	None
DREQ Rcvd		
	Send DREP	CM to TimeWait
	(default)	None
DREQ Sent		
	Timeout	CM to DREP Timeout
	Receive DREQ	CM to DREQ Rcvd
	Receive DREP	CM to TimeWait
	(default)	None
RTU Timeout		
	Retry REP	CM to REP Sent
	No Retry	Send REJ / CM to TimeWait / *CEP to Error*
	(default)	None
TimeWait		
	Entry	CM: Start Timer
	Receive DREQ	CM: Send DREP (if Max CM Retries not exceeded)
	Timer Expiration	CM to IDLE
	(default)	None
DREP Timeout		
	Entry (Retry) (Max Retries not exceeded)	Send DREQ / to DREQ Sent
	Entry (No Retry)	CM to TimeWait
	(default)	None

12.9.8 STATE DETAILS

12.9.8.1 TIMEOUT

A message may be re-sent no more than **REQ**:Max CM Retries , but there is no requirement that it be re-sent that many times.

12.9.8.2 RTU TIMEOUT

If the Passive agent sends **REP** but does not receive either an **RTU** or a message on the CEP (QP or EEC, as appropriate), it transitions to **RTU** Timeout. If it has not exceeded **REQ**:Max CM Retries , the Passive agent may resend **REP**.

12.9.8.3 ESTABLISHED

12.9.8.3.1 REQ RECEIVED / REP RECEIVED

(RC, UC) A CM may receive a **REQ/REP** specifying a remote QPN in "**REQ**:local QPN"/"**REP**:local QPN" that the CM already considers connected to a local QP. A local CM may receive such a **REQ/REP** if its local QP has a stale connection, as described in section 12.4.1. When a CM receives such a **REQ/REP** it shall abort the connection establishment by issuing **REJ** to the **REQ/REP**. It shall then issue **DREQ**, with "**DREQ**:remote QPN" set to the remote QPN from the **REQ/REP**, until **DREP** is received or Max Retries is exceeded, and place the local QP in the TimeWait state.

(RD) A CM may receive a **REQ/REP** specifying a remote EECN in "**REQ**:local EECN"/"**REP**:local EECN" that the CM already considers connected to a local EEC. A local CM may receive such a **REQ/REP** if its local EEC has a stale connection, as described in section 12.4.1. When a CM receives such a **REQ/REP** it shall abort the connection establishment by issuing **REJ** to the **REQ/REP**. It shall then issue **DREQ**, with "**DREQ**:remote EECN" set to the remote EECN from the **REQ/REP**, until **DREP** is received or Max Retries is exceeded, and place the local EEC in the TimeWait state.

If a CM receives a **REQ**/**REP** as described above, if the **REQ**/**REP** has the same Local Communication ID and Remote Communication ID as are present in the existing connection and if the **REQ**/**REP** arrives within the window of time during which the active/passive side could be legally retransmitting **REQ**/**REP**, the CM should treat the **REQ**/**REP** as a retry and not initiate stale connection processing as described above.

12.9.8.4 TIMEWAIT

The **PathRecord:PacketLifeTime** (section 15.2.5.16 PathRecord) field defines the maximum time that a packet can exist in the fabric.

The TimeWait timer shall be set to twice the **PathRecord:PacketLifeTime** value plus the remote's Ack Delay.

The CM is responsible for placing QPs/EECs in the TimeWait state, for maintaining them in that state for a period not less than the TimeWait period, and for removing them afterward.

Receipt of a **DREQ** while in the TimeWait state shall not affect the Time-Wait timer.

12.9.8.5 MESSAGE RECEIPT ACKNOWLEDGMENT (MRA)

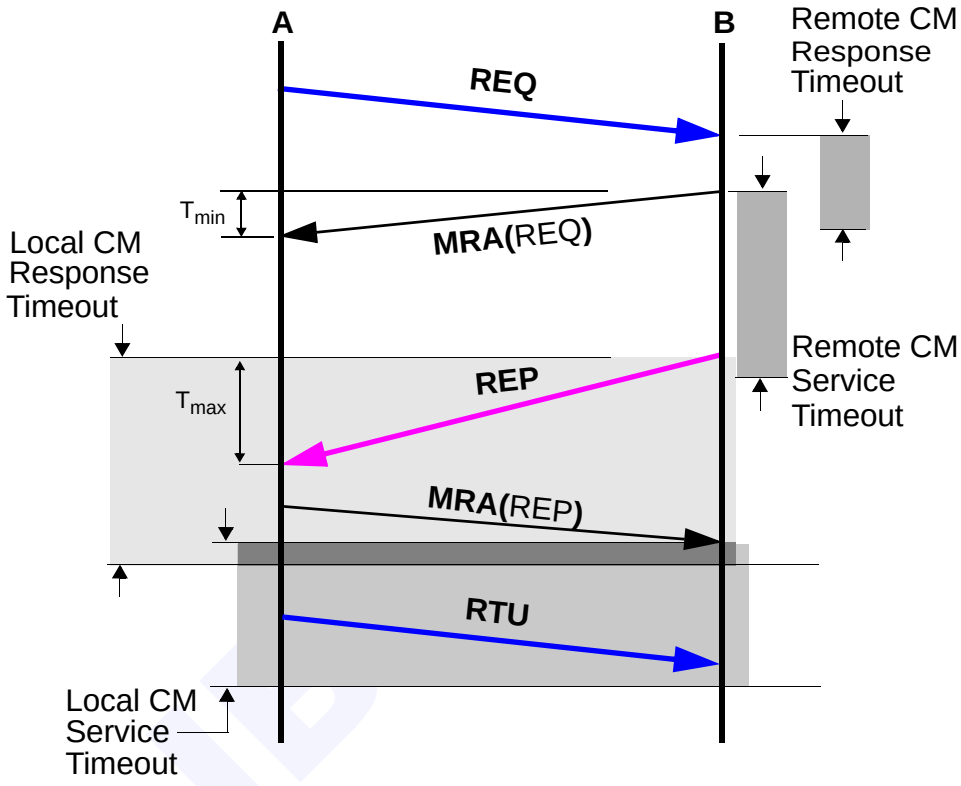


Figure 133 MRA Example

Figure 133 illustrates the use of the **MRA** message in a CM message exchange. 'Local' and 'Remote' are with respect to 'A'. Because B cannot return a **REP** or **REJ** to A within the **Remote CM Response Timeout**, B sends an **MRA(REQ)**, notifying A that B has received the REQ message and is processing it. The **MRA(REQ)** contains B's **CM Service Timeout** value. B completes its processing and sends the **REP** message to A before the expiration of the **Remote CM Service Timeout**.

Because packet flight times may differ due to fabric congestion, (e.g., the **MRA** may travel in the minimum possible time, and the **REP** in the maximum time, as shown by T_{min} and T_{max}), A shall allow an additional PacketLifeTime for the **REP** to arrive.

When A receives the **REP**, it realizes that the required processing will not allow it to transmit a **REJ** or **RTU** soon enough to arrive at B before the **Local CM Response Timeout** expires, so it sends an **MRA(REP)** containing its CM Service Timeout value. When it completes the REP pro-

cessing, A sends the **RTU**, which arrives before the **Local CM Service Timeout** expires.

Once an **MRA** is received, the CM shall not re-send the message acknowledged by the **MRA** sooner than the period of time represented by the applicable CM Service Timeout period plus PacketLifeTime.

12.9.8.6 TIMEOUTS AND RETRIES

In the communication establishment protocol, the sending of the **REQ**, **REP** and **DREQ** messages may be retried by the sender. The retry happens after the sender fails to receive a response message from the recipient within the appropriate response timeout period.

For the **REQ** message, the Remote CM Response Timeout period is the recipient's "turn-around" time. The **REQ** sender may consider the **REQ** (or response) lost after (2*PacketLifeTime + Remote CM Response Timeout). Upon receiving a **REQ,** the recipient must send a **REP**, **REJ**, or **MRA** by the Remote CM Response Timeout. The Service Timeout period begins when the **MRA** is sent, and a **REJ** or **REP** must be sent before it expires.

The Local CM Response Timeout tells the **REP** sender how long to wait for an **MRA**, **REJ**, or **RTU**. The Local CM Response Timeout value includes the round trip flight time. If the **REP** sender receives an **MRA**, it can expect the **REJ** or **RTU** within (Local CM Service Timeout + PacketLifeTime) after the **MRA**'s arrival.

The response timeout period for the **DREQ** message is the Local CM Response Timeout present in the original **REQ** message.

When the sender retries a message send, the recipient can potentially receive multiple copies of the same message. The recipient of a **REQ** (or **REP**) message should determine the amount of time it has to send a response based upon when it received the latest **REQ** (or **REP**) message; the remaining time it has to reply is thus reset back to the full response timeout period each time it receives a new **REQ** (or **REP**) for the same connection establishment attempt.

If the sender of a **REP** message receives another **REQ** message for the same connection establishment attempt, after it resends the **REP** message it should reset its response timeout period back to the full Local Response Timeout period that it received in the **REQ** message.

12.9.8.7 REJ RETRY

If the Active agent receives a **REJ** while it is in the REQ Sent or REP Wait state, the Active agent has the option to perform a retry. If the Active agent chooses not to retry, the CEP is put into the Error state and the Active

agent proceeds to the Idle state. If a retry is performed, the Active agent must re-send the **REQ** message using a different Local Communication ID than was used for the original **REQ** message that received the **REJ** in reply.

12.9.8.8 REJ SENT

When the Passive agent sends a **REJ**, it enters the REJ Sent state. If the Passive agent chooses not to retry its connection establishment attempt, it places the CEP into the Error state, which if the CEP is a Queue Pair will cause any Work Requests posted to the Receive Queue of the Queue Pair to be flushed. If the Passive agent wishes to avoid having its Work Requests flushed, it may instead stay in the REJ Sent state until is receives a **REQ**, at which time it proceeds to the REQ Rcvd state and connection establishment proceeds normally.

12.9.8.9 REP SENT / MRA(REP) RECEIVED

If the Passive agent receives a **LAP** for a connection while it is in the REP Sent or MRA(REP) Rcvd state, when/if the Passive agent times out and enters the RTU Timeout state it should take the “Retry REP” transition out of that state (assuming it has not exceeded the retry count specified in the **REQ** message). Receiving a **LAP** is a strong indication that the Active agent believes the connection has been established. By retrying the **REP**, the Passive agent informs the Active agent that the connection has in fact not been fully established, which will prompt the Active agent to retry the **RTU** so that it can be fully established. If the Passive agent does not retry the **REP**, it is possible that the Active agent will get stuck in a loop retrying its **LAP** message until the Passive agent ultimately times out the connection establishment attempt and sends a **REJ**.

12.9.9 CONNECTION STATE

Communication Managers shall maintain the following information for the life of a connection:

- Local Communication ID
- Remote Communication ID
- Local CM Response Timeout
- Remote CM Response Timeout
- Local QPN / Local EECN
- Remote CA QPN / Remote CA EECN
- Remote CA GUID
- Remote CM LID / Remote CM GID
- Max CM Retries
- ClassVersion, if multiple ClassVersions are supported

12.10 COMMUNICATION ESTABLISHMENT LADDER DIAGRAMS

The following ladder diagrams show the message exchanges for various communication establishment scenarios. These are not applicable to Unreliable Datagrams (see section 12.11 for Service ID Resolution).

12.10.1 ACTIVE CLIENT TO PASSIVE SERVER - BOTH CLIENT AND SERVER ACCEPT COMMUNICATION

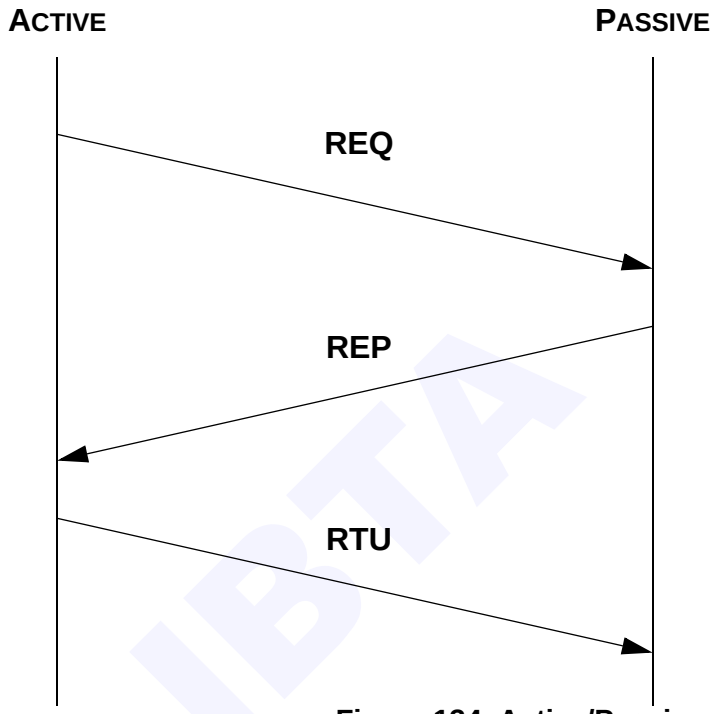


Figure 134 Active/Passive, Both Accept

For RC and UC service, the above exchange establishes the connection.

For RD service, the above exchange must be performed

- To establish a Reliable Datagram Channel between two EECs
- To resolve a Service ID and associate QPs for possible use over an existing RDC

How cooperating applications exchange information on additional available QPs is specific to the applications.

12.10.2 ACTIVE CLIENT TO PASSIVE SERVER - SERVER REJECTS COMMUNICATION

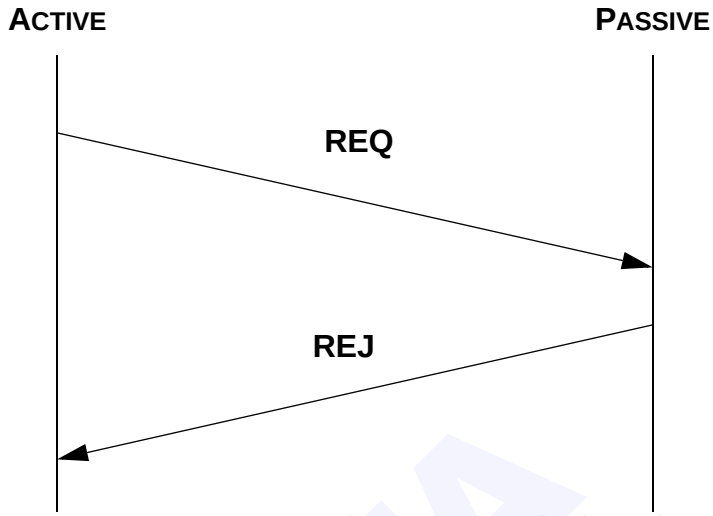


Figure 135 Active/Passive, Server Reject

The above exchange occurs when the passive server cannot or will not perform the requested action. The **REJ** message contains the reason why the action was not performed.

12.10.3 ACTIVE CLIENT TO PASSIVE SERVER - CLIENT REJECTS COMMUNICATION

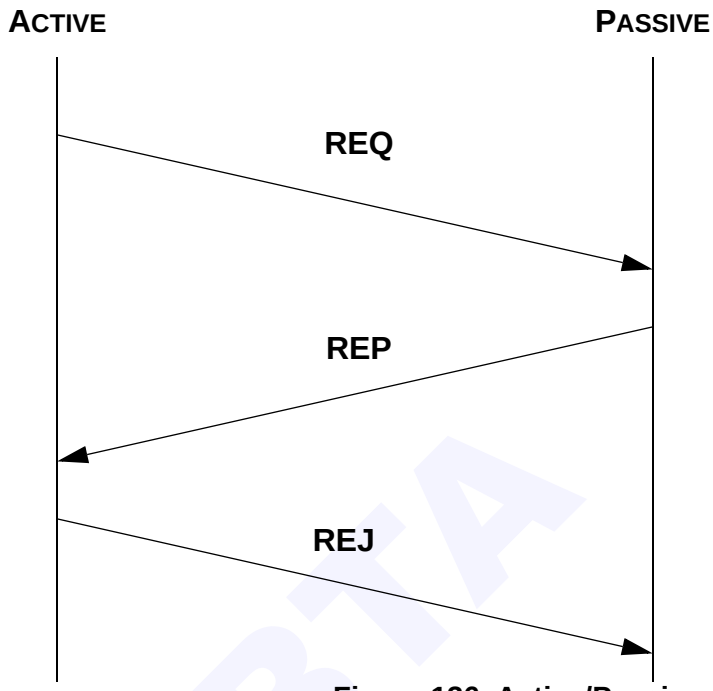


Figure 136 Active/Passive, Client Reject

The above exchange occurs when the requesting client decides not to continue with the requested action. (An example is a client that requires Automatic Path Migration support not provided by the server.) The **REJ** message contains the reason why the action was not continued.

12.10.4 PEER TO PEER - BOTH ACCEPT COMMUNICATION

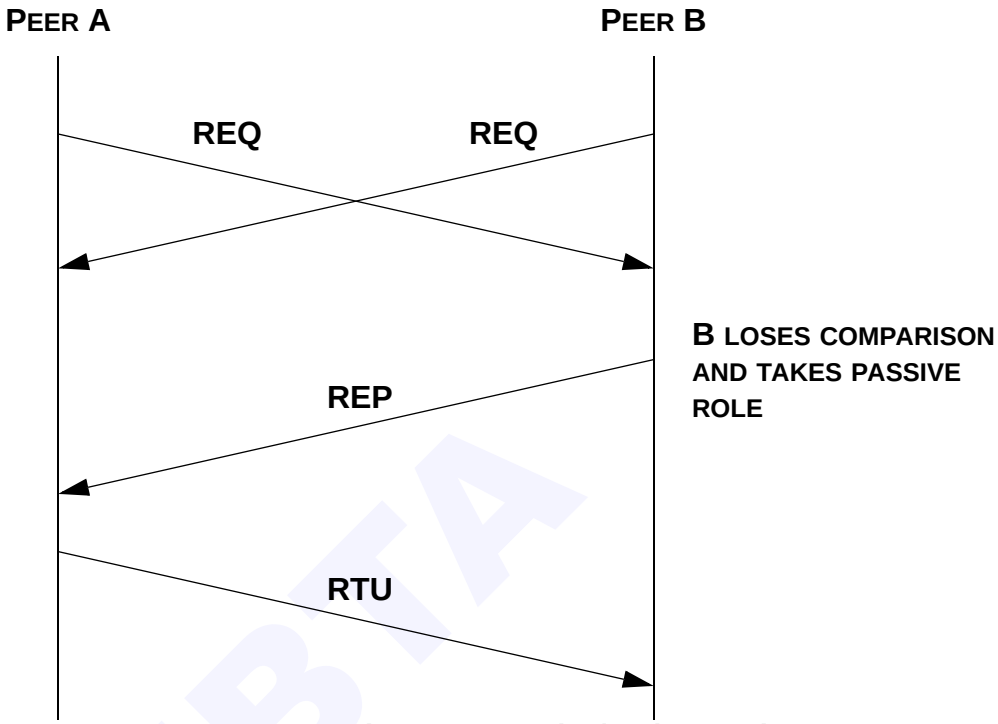


Figure 137 Active/Active, Both Accept

The above exchange occurs when two peer entities attempt communication. In this case, the ServiceID in both **REQ** messages is the same. The ServiceID implicitly defines whether the service is client/server or peer to peer, but the application must inform the CM so that the CM will handle the inbound **REQ** correctly. (The CM protocol state transitions are different for a peer to peer communication establishment attempt than for a client/server communication establishment attempt. In particular, the Peer Compare state is only entered for a peer to peer communication establishment attempt. See section 12.9.7.1 for a description of the Peer Compare state.)

Peer A and Peer B compare their CA (Channel Adapter) GUIDs, treating each as a big-endian value, to decide which party will take the active side of the CM protocol. The peer with the numerically smaller GUID assumes the passive role in the remainder of the communication establishment protocol.

If the CA GUIDs match (e.g., two processes using the same CA), the peers compare their QPNs (using the **REQ:Local QPN** field) in the case of connection establishment between Connected Queue Pairs, or their EECNs (using the **REQ:Local EECN** field) in the case of Reliable Data-

gram Channel establishment. The comparison is done treating each as a big-endian value, with the smaller QPN/EECN taking the passive role.

CM shall not be used to establish “loopback” channels on a single QP.

12.10.5 ACTIVE PEER TO ACTIVE PEER - PASSIVE REJECTS COMMUNICATION

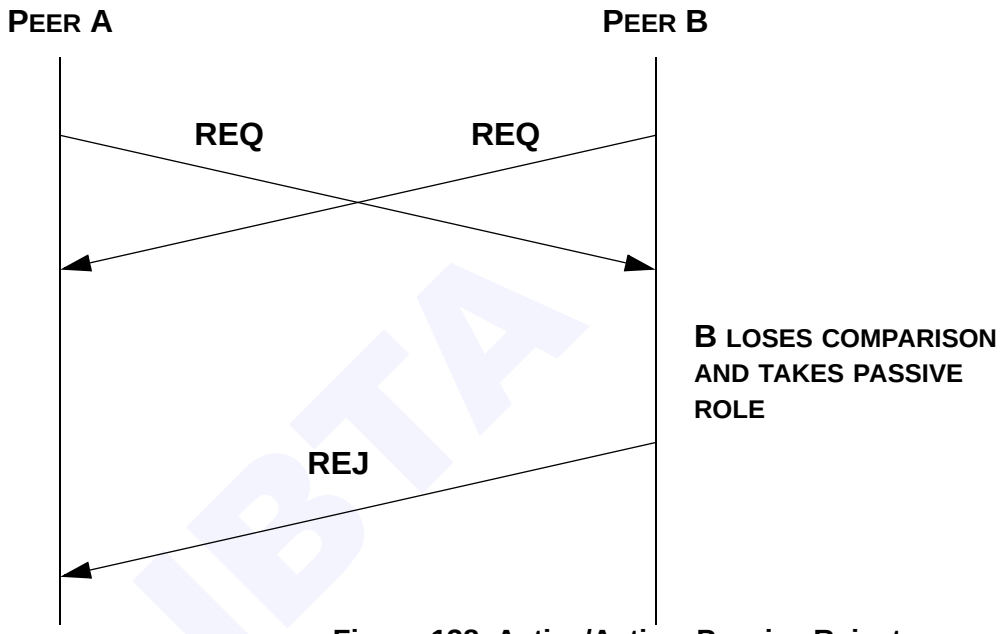


Figure 138 Active/Active, Passive Reject

The above exchange occurs when the ’losing’ peer decides not to continue the requested action. The **REJ** message contains the reason the action was not continued.

12.10.6 ACTIVE PEER TO ACTIVE PEER - ACTIVE REJECTS COMMUNICATION

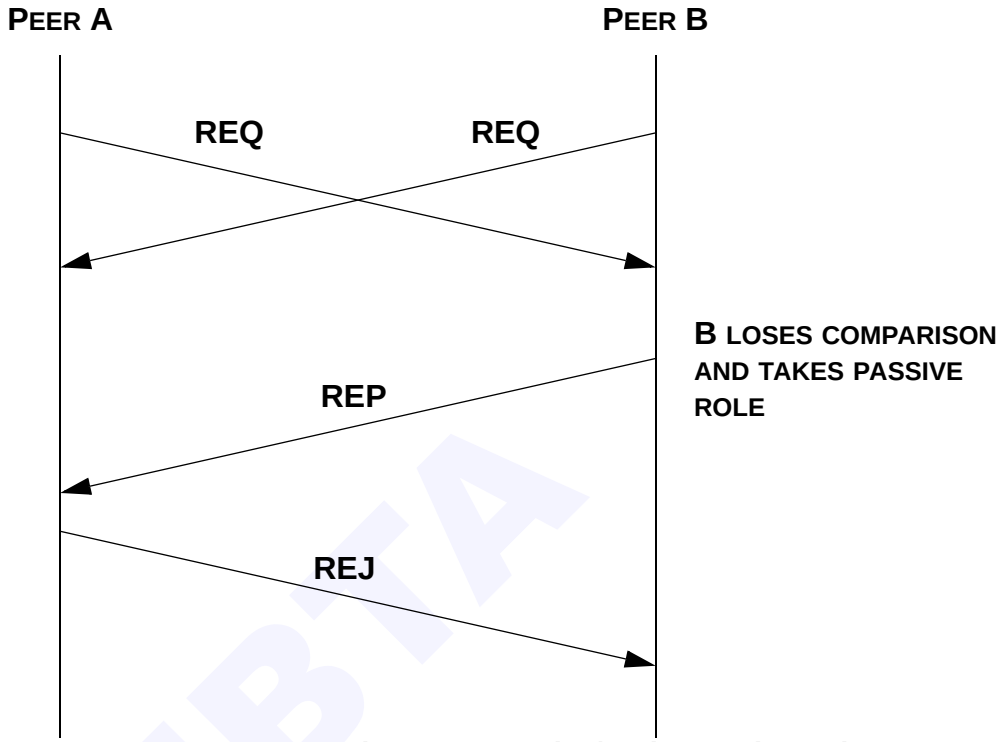


Figure 139 Active/Active, Active Reject

The above exchange occurs when the when the 'winning' peer decides not to continue with the requested action. The **REJ** message contains the reason why the action was not continued. The peer receiving the **REJ** message returns to the IDLE state.

12.10.7 ACTIVE CLIENT TO PASSIVE SERVER WITH REDIRECTOR - ALL ACCEPT COMMUNICATION

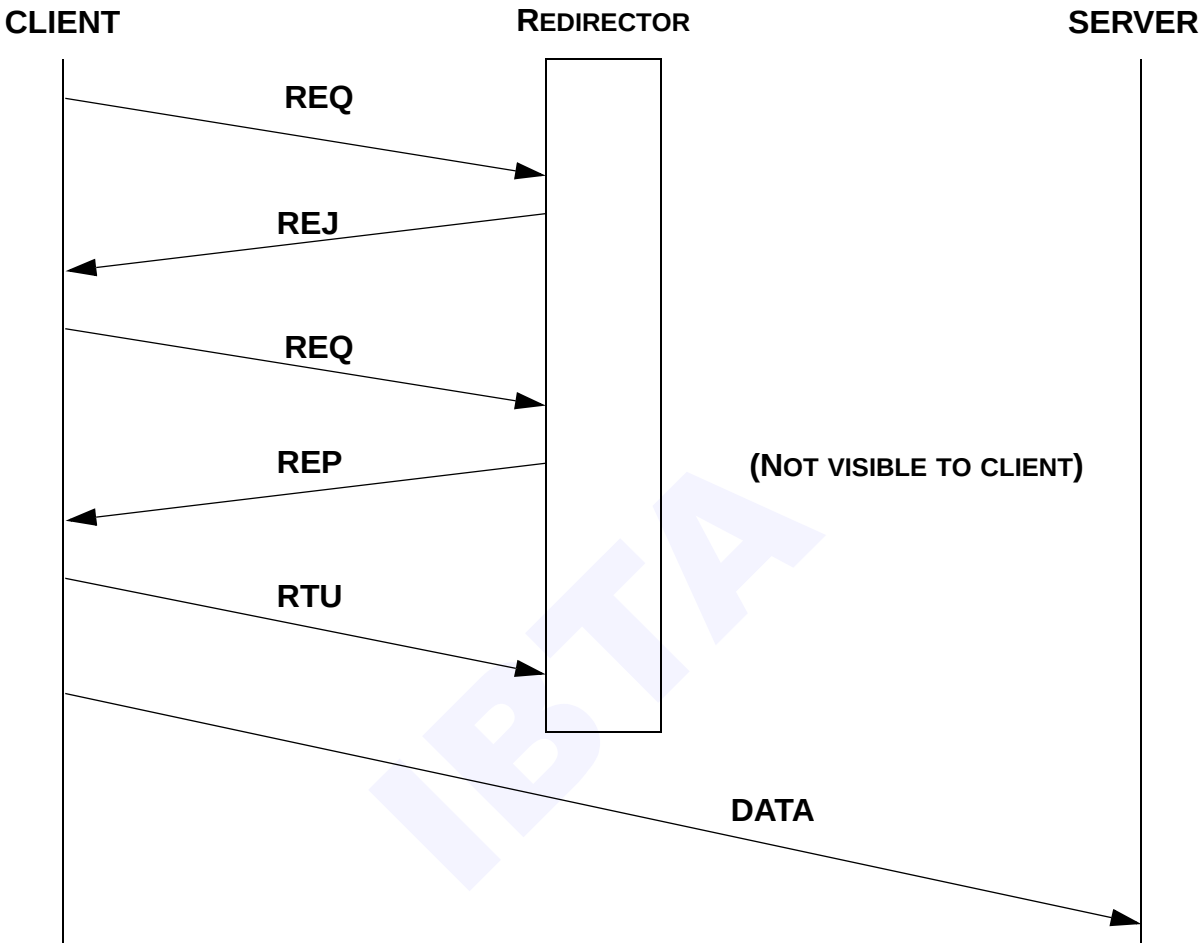


Figure 140 Redirection, Accepted

A redirector is a CM that provides CM services on behalf of an entity supported on a port other than the redirector CM's port. The port information for both endpoints is explicit in the **REQ** and **REP** messages, allowing the redirector to manage connections as a proxy for another entity.

The above exchange occurs when the redirector sends **REJ** with the Status "Port Redirection", indicating that the requested ServiceID is available at a different port. The requesting client (using a new Local Communication ID) sends a new **REQ** proposing the port specified in the **REJ**. All CM messages are exchanged between the client CM and the redirector, but traffic over the established connection goes between the client and the server.

12.10.8 COMMUNICATION RELEASE

The following ladder diagram shows the message exchange for communication release.

Communication release as illustrated in this section is ungraceful. Upon receipt of a Disconnect Request, each CM shall cause the affected QP to be placed into the error state, causing pending work requests to complete with the Flush error status.

Consumers are free to define and execute a more graceful communication release protocol that allows for an orderly shutdown of communications. Any such protocol shall utilize the communication release protocol illustrated below after the termination of normal message processing.

12.10.8.1 DISCONNECT REQUEST

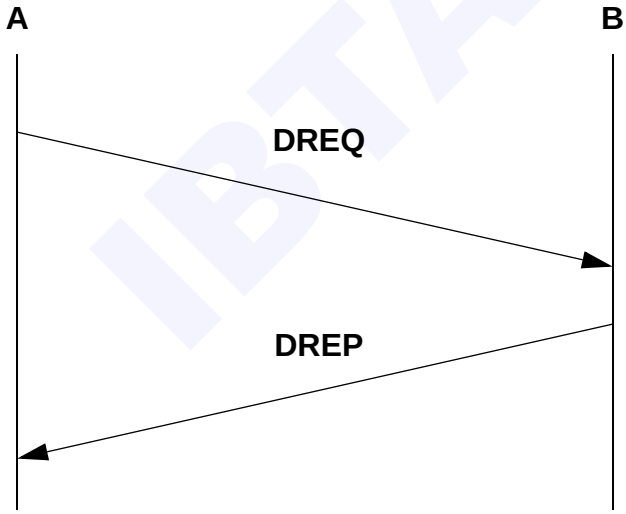


Figure 141 Disconnect Request

Because the **DREQ** and **DREP** travel out of band relative to normal communications traffic, how operations currently in progress will be completed cannot be predicted.

12.11 SERVICE ID RESOLUTION PROTOCOL

Service ID Resolution (SIDR) provides a way for users of Unreliable Datagram (UD) service to determine a Queue Pair on the target port that supports a given Service ID.

GSAs shall support this protocol if non-management services are provided on the Channel Adapter at other than fixed QPNs. If this protocol is

not supported by a CM, the IsSIDRCapable bit shall be reset in the CapabilityMask component of the ClassPortInfo attribute for the CM. See 16.7.3 Attributes for details.

The protocol consists of a single request and a single reply, using unreliable Management Datagrams (MADs) targeted to the GSI. SIDR messages are of the Communication Management class.

If the SIDR response returns a valid QPN, the returned QPN shall be in the partition identified by the P_Key in the header of the SIDR Request.

12.11.1 SIDR_REQ - SERVICE ID RESOLUTION REQUEST

SIDR_REQ requests that the recipient return the information necessary to communicate via UD messages with the entity specified by **SIDR_REQ:ServiceID**.

Table 110 SIDR_REQ Message Contents

Field	Description	Byte[Bit] Offset	Size, bits (Values)
RequestID	See section 12.11.1.1	0	32
Partition Key	See section 12.11.1.2	4	16
(reserved)		6	16
ServiceID	See section 12.11.1.3	8	64
Private Data	See section 12.11.1.4	16	1728

12.11.1.1 REQUESTID

A 32-bit value identifying the request. The target of the request shall return this value unchanged in the **SIDR_REP** message. If a **SIDR_REQ** message is re-sent, the sender shall send the same RequestID. The SIDR_REQ recipient shall use the Source GID (or Source LID, if GRH not present) to distinguish between requests from different requestors that might have the same RequestID.

12.11.1.2 PARTITION KEY

The Partition Key that the sender wishes to use to access the service identified by Service ID.

12.11.1.3 SERVICE ID

The Service ID for which the sender wishes the responder to provide UD addressing information. See section 12.7.3 for more information.

12.11.1.4 PRIVATE DATA

Data that is opaque to the SIDR protocol for use by the requester and responder. For example, some systems may require that the PrivateData

area contain an authorization key before reporting the QP supporting certain ServiceIDs.

12.11.2 SIDR_REP - SERVICE ID RESOLUTION RESPONSE

SIDR_REP returns the information necessary to communicate via UD messages with the entity specified by **SIDR_REQ:ServiceID**.

Table 111 SIDR_REP Message Contents

Field	Description	Byte [Bit] Offset	Length, bits
RequestID	See section 12.11.1.1	0	32
Status	See section 12.11.2.1	4	8
Additional Information Length	If non-zero, the length in bytes of valid Additional Information. The sender is not required to provide Additional Information even if the Status code it places in the SIDR_REP message allows for it. If the sender decides not to provide Additional Information, it shall set this field to a value of zero.	5	8
(reserved)		6	16
QPN	See section 12.11.2.2	8	24
(reserved)		11	8
ServiceID	See section 12.11.1.3.	12	64
Q_Key	See section 12.11.2.3	20	32
Additional Information	See section 12.11.2.1	24	576
Private Data	See section 12.11.1.4	96	1088

12.11.2.1 STATUS

The Status field tells whether the QPN and Q_Key fields are valid, and if not valid, the reason a valid QPN and Q_Key were not provided. Depending upon the value of the Additional Information Length field, the Additional Information field may or may not have valid contents.

Value	Description	Meaning of Additional Information Field (when present)
0	QPN and Q_Key are valid	
1	Service ID not supported	
2	Rejected by Service Provider	

3	No QP available	
4	Request should be redirected to the endpoint specified by ClassPortInfo. QPN and Q_Key are not valid.	This is a ClassPortInfo data structure as documented in section 13.4.8.1, describing where to redirect the **SIDR_REQ**.
5	Class Version specified in the header for the **SIDR_REQ** MAD is not supported. When returning this code, the "Code for Invalid Field" portion of the Status field in the MAD header associated with this **SIDR_REP** message should contain a value of 1. (The class version specified is not supported.) See Table 115 MAD Common Status Field Bit Values for details.	Highest CM Class Version that is supported.
6-255	Reserved	

12.11.2.2 QPN

The QPN of the local QP on which the requested Service ID is supported. (Only valid if so indicated by Status field).

12.11.2.3 Q_KEY

The Q_Key for the QP returned in **QPN**.

12.11.3 PATH INFORMATION

The information returned in the **SIDR_REP** message is insufficient, by itself, to create a usable address handle. Specifically, the values for **PathRecord:Mtu** and **PathRecord:Rate** are required except when sending packet payloads no larger than the minimum PMTU, or when transmitting on a minimum-width link, respectively. These values are available through Subnet Administration (see section 15.2.5.16).

CHAPTER 13: MANAGEMENT MODEL

13.1 INTRODUCTION

IBA management is built on top of four fundamental concepts. These include:

- management entities,
- agents,
- a messaging scheme,
- a collection of specific messages including message content, and related behaviors.

An agent is a conceptualization of a body of low level functionality embedded in all channel adapters, switches, and routers, which provides the means to set and query various parameters internal to the channel adapter, switch, or router.

Managers and interested parties are conceptualizations of high level bodies of functionality which provide for controlling or examining various aspects of subnet or fabric configuration and operation.

The messaging scheme provides for intercommunication between managers or interested parties and agents, and, in some cases, between agents. The messaging scheme specifies the basic message types and interfaces through which agents and managers exchange information.

Finally, specific messages and message sequences are defined in terms of message content and associated required behaviors. Messages are grouped into classes according to the type of management activity the messages support.

The specification of management operations is done from the viewpoint of specifying messages that may appear on the wire and specifying behaviors associated with those messages. The appearance of a message at a port implies a required action and, possibly, response. Additionally, the appearance of a message on the wire implies behavior of the entity that caused the message to be emitted. In particular, the behavior requirements in certain areas (e.g. subnet management, see 14.4 Subnet Manager on page 859) imply the existence of certain entities (e.g. a subnet manager) which embody required behaviors with respect to the origination and consumption of various messages.

Various conceptualizations are used in specifying behaviors. However, the use of such conceptualizations in this and other management related chapters is purely a descriptive artifice. The conceptualizations themselves, do not convey normative requirements. Normative requirement specification is done by, and only by, specification of message formats and associated required behaviors. Finally, while some conceptualizations may suggest certain implementations, implementations are outside of the scope of the specification and no specific implementation is implied.

13.2 ASSUMPTIONS, AND SCOPE

13.2.1 ASSUMPTIONS

There are certain assumptions that underlie the management mechanisms specified herein. Proper operation of the management mechanisms and fulfillment of the objectives underlying these specifications is predicated upon the validity of these assumptions. While the assumptions themselves are not part of the specification, they are an essential element of the framework in which these specifications apply.

- The management operations specified herein provide for a level of interoperability such that an SM from any vendor can manage a heterogeneous collection of IBA-compliant channel adapters, switches, and routers from any set of vendors. However, compatibility and interoperability among SMs from different vendors is not supported. Migration from one vendor's SM to another's by way of system reinitialization, i.e., through a planned outage, is supported. Such migration assumes appropriate steps of transferring data between vendors' SMs have been accomplished prior to the re initialization.
- The management operations specified herein provide the means to conduct a variety of activities. Some of the mechanisms specified are optional. And, except as specifically stated, the specification of a means does not imply or require that the means be used. It is assumed that each fabric will be constructed, configured, and operated according to the needs of its user(s) and that constructors exercise diligence in selection of components to ensure the fabric possesses the characteristics required. For example, if a user requires multicast support but mixes components that do and do not support multicast, they may fail to achieve their requirements.

13.2.2 SCOPE

As noted above, a number of management classes are distinguished in the IBA management model. The classes include:

- Subnet management. Subnet management is the body of activity associated with discovering, initializing, and maintaining an IBA subnet. In addition, the subnet management sections specify methods for interfacing to a diagnostic framework for handling subnet and protocol errors. (See 14.2.5.14 VendorDiag on page

840). In the following sections a subnet manager will be denoted by SM while a subnet management agent will be denoted by SMA.

- Subnet administration (SA): Subnet administration provides a means for management entities and applications to obtain information about fabric configuration and operation. (See Chapter 15: Subnet Administration on page 882). In the following sections subnet administration will be denoted by SA.
- Communication management: Communication management provides the means to set up and manage communications between a pair of queue pairs or, in certain cases, to identify which queue pair to use for a certain service. (See Chapter 12: Communication Management on page 650 and 16.7 Communication Management on page 1011). In the following sections a communications manager will be denoted by CM.
- Performance management: Performance management specifies a set of facilities for examining various performance characteristics of a fabric. (See 16.1 Performance Management on page 930).
- Device management: Device management specifies the means for determining the kind and location of various kinds of devices on a fabric. (See 16.3 Device Management on page 985).
- Baseboard management: Baseboard management specifies the means to effect, in-band (i.e. over the IBA fabric) low level system management operations. (See 16.2 Baseboard Management on page 973 and InfiniBand Architecture Specification Volume 2, Chapter 13, Hardware Management).
- SNMP tunneling: SNMP tunneling specifies mechanisms to support transport of SNMP operations through an IBA fabric. (See 16.4 SNMP Tunneling on page 998).
- Vendor specific: The vendor specific classes specify a basic framework within which a vendor can define vendor specific management communications and operations that are beyond the scope of the IBA. See 16.5 Vendor-specific on page 1005 for architectural details relating to use of specific values.
- Application specific: The application specific classes specify a basic framework within which services can be defined which implement operations that are beyond the scope of the IBA. See 16.6 Application-specific on page 1008 for architectural details relating to use of specific values.

As a notational convenience, the set of classes as listed above but excluding subnet management are referred to as General Services. When referring to General Services Managers, the notation GSM may be used. When referring to General Services Agents, the notation GSA may be

used. According to the context in which it appears, GSM(s) or GSA(s) may refer to the group of all supported general services managers or agents on a channel adapter, switch, or router, or to any of the managers or agents in that group.

The IBA management services provided by the above classes support management of only the devices that comprise the IBA subnet. They do not support management of devices beyond the subnet. Specifically, they do not support management of tape drives, hard disk drives, network interfaces, etc. Mechanisms required to discover and power-manage devices that are accessed through an IBA channel adapter are provided within the above classes but provision of services specific to such devices is beyond the scope of the IBA.

IBA management provides a means of configuring and gathering information from IBA channel adapters, switches, and routers.The IBA Subnet Administration Service provides a means for other entities to determine the topology and configuration of the subnet. For example, operating systems or other higher level management entities may use IBA Subnet Administration services mechanisms to enforce operating system policies, or cluster policies, and so on, but such higher level entities and the policies they effect are outside the scope of IBA management services.

A variety of standards for communication of management information between managed elements and management applications exist today. These include Simple Network Management Protocol (SNMP), Desktop Management Interface (DMI), and Common Information Model (CIM), as well as other standard and proprietary interfaces. Such standards may be layered on top of the IBA management model interfacing to it through services defined in the model. Alternatively, they may interface to IBA management elements through private interfaces. In either case, while the IBA management model provides means for such applications to obtain subnet topology and configuration information, such applications are outside the scope of IBA management.

Finally, the current IBA specification defines only the mechanisms required for proper operation of IBA fabrics and interoperation of IBA components. Specific applications, such as for enclosure management, can also be used in conjunction with non-IBA subsystems that are connected to the IBA subnet. Such applications may utilize the IBA subnet as a means of transport for the specific subsystem management data but such subsystem management services themselves are outside of the scope of the management services specified for IBA.

Figure 142 Management Model on page 713, below, depicts an example subnet indicating graphically the relationships among the IBA managed

subnet and related services and higher level and lower level entities that may be found on an IBA fabric.

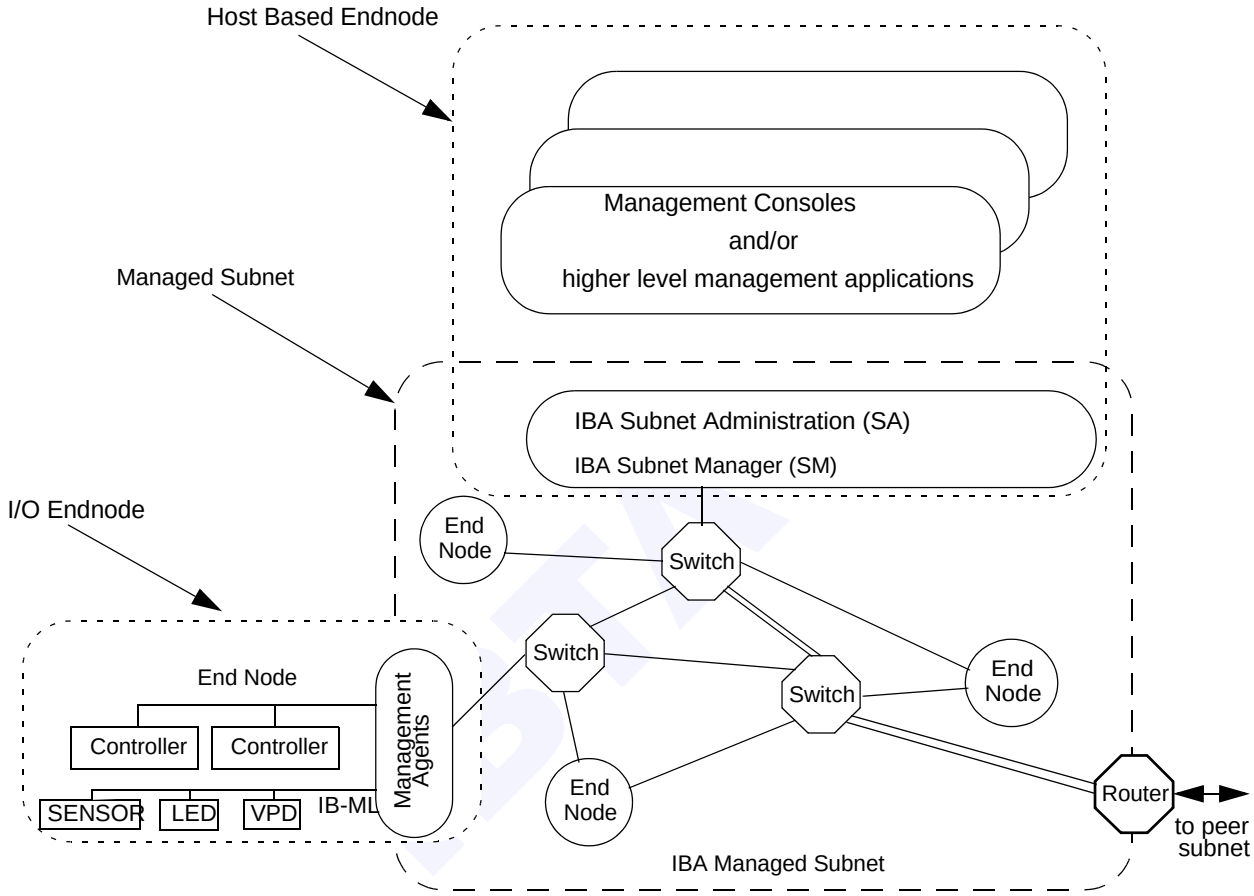


Figure 142 Management Model

This chapter provides an overview of the IBA management model, the management entities, and the corresponding interfaces. In addition this chapter defines requirements and specifies mechanisms common to all management activities. Subsequent chapters specify additional details associated with specific management classes. For each management class, the complete set of applicable requirements that must be satisfied and mechanisms that must be provided is the combination of those from this chapter with those in the corresponding class specific sections of the other chapters.

13.3 MANAGERS, AGENTS, AND INTERFACES

13.3.1 INTRODUCTION

IBA Management is organized around abstract functional entities referred to as *managers*, *agents,* and *interfaces*. Communication between managers and agents is performed through management messages referred

to as Management Datagrams (MADs). MADs are exchanged using the unreliable datagram transport service as defined in 9.8.3 Unreliable Datagrams on page 389.

Managers are conceptual functional entities that effect control over fabric elements or provide for gathering information from fabric elements. In general, managers may reside anywhere in a subnet although class specific constraints on the manner in which they logically interface to the fabric medium (e.g. SMs use QP0, see 13.5.1 MAD Interfaces on page 749) may impose specific restrictions.

Agents are conceptual functional entities present in IBA channel adapters, switches, and routers that process management messages arriving at the ports of the IBA channel adapters, routers, and switches where they exist.The functionality represented by an agent effects required behaviors associated with MADs which arrive at the port or ports with which it is associated.

Abstractly, interfaces represent a target to which messages may be sent and through which messages will be processed or will be dispatched to an appropriate processing entity. For management interfaces, the associated processing entity is an agent or, in some cases, a manager. As such, an interface is a means to gain access to the functionality of agents and/or managers.

Management operations are divided into a set of classes. For a given class of activity, there is usually only a small number of managers on a subnet. Conceptually, for each supported class, there is one agent on each switch, channel adapter, and router on the IBA subnet.

Although the notions of agent, manager, and interface as described above, may suggest specific implementations, this specification only mandates behavior with respect to sourcing and sinking management messages, not how that behavior is achieved. The notion of an agent, manager, or interface, is a convenient descriptive artifice which encapsulates functional operations and behaviors associated with a particular class of activities. This specification does not require the existence of agents, managers, or interfaces per se. It does require that implementations exhibit the behaviors associated with the abstract agents, managers, and interfaces. How that is actually accomplished is implementation dependent.

The messages and behaviors relating to the subnet management class are further defined in 14.2 Subnet Management Class on page 794. This class uses specialized MADs referred to as Subnet Management Packets (SMPs).

Figure 143 Typical Subnet Manager/Agent Relationships on page 715 depicts a single subnet showing representative relationships among channel adapters, switches, subnet managers and agents.

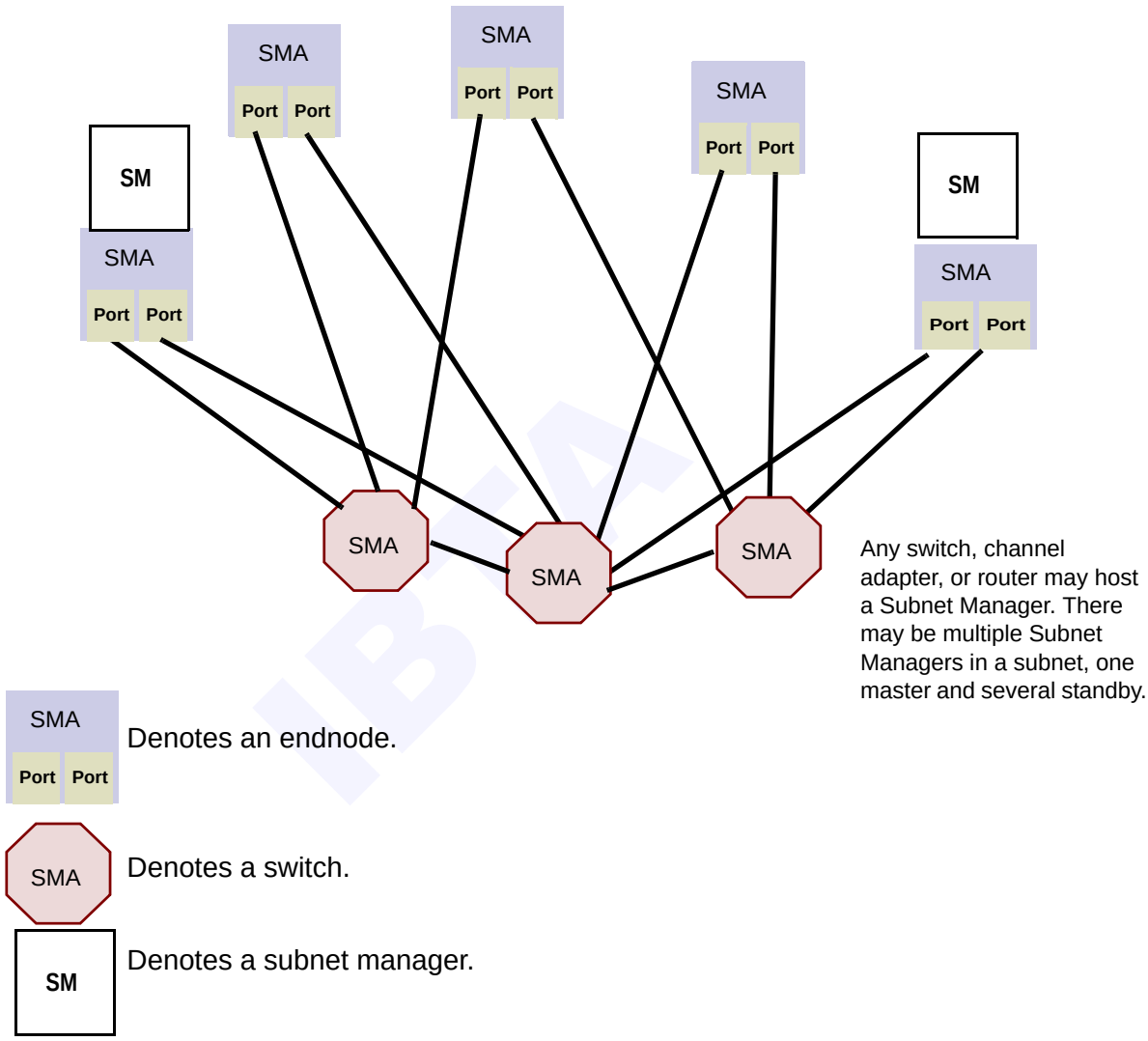


Figure 143 Typical Subnet Manager/Agent Relationships

The messages and behaviors relating to the subnet administration class are further defined in 15.2 SA MADs on page 883 while the messages and behaviors relating to the other general services classes are further defined in the subsections of Chapter 16: General Services on page 930. These service classes use MADs referred to as General Services Management Packets (GMPs).

Figure 144 Typical General Services Management/Agent Relationship on page 716 depicts a single subnet showing representative relationships between general service class managers and corresponding agents.

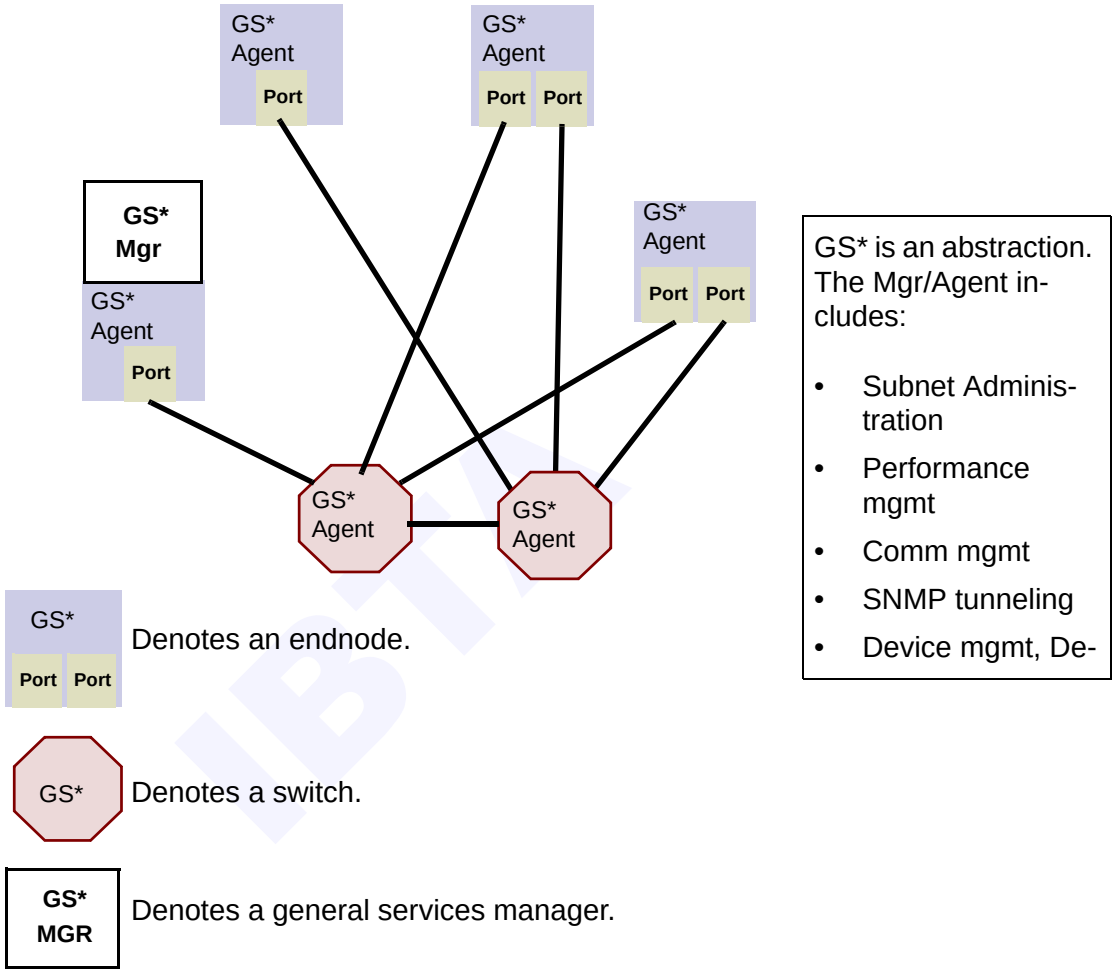


Figure 144 Typical General Services Management/Agent Relationship

13.3.2 REQUIRED MANAGERS AND AGENTS

C13-1: Each subnet **shall** have at least one logical SM.

Logical SMs may be single physical entities or may consist of multiple, possibly distributed, cooperating physical entities which collectively effect the appearance of a single SM to CAs, switches and routers on the subnet it manages.

If there is more than one entity capable of acting as a master SM, only one should function as a master SM during initialization.

See Chapter 14: Subnet Management on page 794 for additional specific requirements applicable to SMs during and after initialization.

There is a close relationship between SMs and SAs. This is described in 15.1.2 Relationship Between SA and the SM on page 883.

IBA does not otherwise mandate the existence of, the location of, or, operational characteristics of GSMs. The class specific sections of Chapter 16: General Services on page 930 define messages and agent behaviors available that GSMs depend on but there are no manager specific messages or related behaviors that GSMs must support.

C13-1.1.1: Every IBA compliant channel adapter, switch, or router **shall** support the functionality characterized as a Subnet Management Agent (SMA).

Supporting the functionality characterized as an SMA means that the channel adapter, switch, or router conforms to the compliance statements for the SMA specified in Chapter 14: Subnet Management on page 794. The specific requirements for supporting this functionality at the various ports of the device are specified in the chapter covering the specific type of device. See Chapter 17: Channel Adapters on page 1016, Chapter 18: Switches on page 1040 and Chapter 19: Routers on page 1059.

Every IBA compliant channel adapter, switch, or router supports the functionality characterized as the various GSAs for those general services specified to be mandatory in the class specific section of Chapter 16: General Services on page 930. Supporting the functionality characterized as a GSA means that the channel adapter, switch, or router conforms to the compliance statements for the GSA specified in the corresponding class-specific sections of Chapter 16: General Services on page 930. The specific requirements for supporting this functionality at the various ports of the device are specified in the chapter covering the specific type of device. See Chapter 17: Channel Adapters on page 1016, Chapter 18: Switches on page 1040 and Chapter 19: Routers on page 1059.

13.4 MANAGEMENT DATAGRAMS

Management Datagrams (MADs) are the basic elements of the messaging scheme defined for management communications. MADs are classified into predefined management classes and for each MAD there is a specified format, use, and behavior. This section specifies characteristics, i.e. formats and associated behaviors, common to all MADs or common across multiple classes. MADs specific to a class are specified in class specific sections of Chapter 14: Subnet Management on page 794, Chapter 15: Subnet Administration on page 882, and Chapter 16: General Services on page 930.

13.4.1 CONVENTIONS

C13-2: For all MADs, for both the fields in the MAD header as well as the fields in MAD attributes, bit placement follows the conventions specified

in 1.5 Document Conventions on page 66. In addition, the following conventions **shall** be observed.

- Fields within a MAD may be either fixed length or variable length within a fixed length location. A variable length field placed in a fixed length location is placed in the high order bits of the fixed length location and the remainder of that location is filled with zero.
- Reserved fields **shall** be filled with 0 by the requester and ignored by the receiver.
- When constructing a response MAD that contains all or part of the corresponding request MAD, it is acceptable to include the contents of reserved fields in the request MAD in the response MAD without regard to their content. That is, such fields need not be set to zero in the response MAD.
- In attribute descriptions in subsequent sections, fields specified as read only (RO) are not alterable by means of MADs. The mechanisms for setting such fields are implementation dependent and outside of the scope of the IBA. With respect to MADs that set values, recipients **shall** ignore any bits in the attribute in a request that correspond to RO components of the attribute being set.
- In attribute descriptions in subsequent sections, fields specified as read write (RW) are settable by means of MADs.

When the term “GID” is used in all attribute descriptions and other text throughout chapters Chapter 13: Management Model on page 709, Chapter 14: Subnet Management on page 794, Chapter 15: Subnet Administration on page 882, and Chapter 16: General Services on page 930, it refers only to a unicast GID unless otherwise specified.

13.4.2 MANAGEMENT DATAGRAM FORMAT

C13-3: The data payload (as used in Chapter 9: Transport Layer on page 230) for all MADs **shall** be exactly 256 bytes.

C13-4: The data payload **shall** include, and only include, the items defined in the MAD base format in Figure 145 MAD Base Format on page 719, with semantics as described in Table 112 Common MAD Fields on page 719.

This includes the offset and alignment within that data area of any of the common attributes specified in 13.4.8 Management Class Attributes on page 732, which may vary by management class.

All MADs consist of a MAD header and MAD data. Except as noted, the MAD header definition is the same for all MADs. The contents of MAD

data areas vary by management class and the specific attribute within the class

bytes	bits 31-24	bits 23-16	bits 15-8	bits 7-0	
0	BaseVersion	MgmtClass	ClassVersion	R	Method
4	Status		ClassSpecific		
8	TransactionID				
12					
16	AttributeID		Reserved		
20	AttributeModifier				
24	Data				
...					
252					

Figure 145 MAD Base Format

13.4.3 MANAGEMENT DATAGRAM FIELDS

Table 112 Common MAD Fields on page 719 lists fields that are common to all MADs. Each class may specify additional class specific usage for certain of these fields.

Table 112 Common MAD Fields

Field Name	Length (bits)	Offset (bits)	Description
BaseVersion	8	0	Version of MAD base format. This **shall** be 1.
MgmtClass	8	8	Class of operation. See Table 113 Management Class Values on page 720 for definition and use.
ClassVersion	8	16	Version of MAD class-specific format. This **shall** be 1 unless otherwise specified in the relevant IBTA management class-specific sections of the specification.
R	1	24	Response bit. See 13.4.5 Management Class Methods on page 721 for definition and usage.
Method	7	25	Method to perform based on the management class. See 13.4.5 Management Class Methods on page 721 for definition and usage.
Status	16	32	Code indicating status of operation. See 13.4.7 Status Field on page 731 for definition and usage.
ClassSpecific	16	48	This field is reserved except for the Subnet Management class. See 14.2.1.2 SMP Data Format - Directed Route on page 796 for definition and usage for Subnet Management.
TransactionID	64	64	Transaction identifier. See 13.4.6.4 TransactionID usage on page 731. This field, if unused by the management class, **shall** be set to 0.

Table 112 Common MAD Fields (Continued)

Field Name	Length (bits)	Offset (bits)	Description
AttributeID	16	128	Defines objects being operated on by a management class. This field, if unused, **shall** be set to 0. See 13.4.8 Management Class Attributes on page 732 as well as class specific sections of Chapter 14: on page 794, Chapter 15: on page 882, and Chapter 16: on page 930 for definition and usage.
Reserved	16	144	Reserved
AttributeModifier	32	160	Provides further scope to the attributes. Usage is determined by the management class and attribute. This field, when not used for the combination of management class and attribute specified in the header, **shall** be set to 0.
Data	1856	192	The data area, usage is defined within the scope of the management class.

13.4.4 MANAGEMENT CLASSES

C13-5: MADHeader:MgmtClass **shall** be one of the values defined in Table 113 Management Class Values on page 720 not defined as reserved.

The functionality provided by specific classes is specified in Chapter 13: Management Model on page 709, Chapter 13: Management Model on page 709, and Chapter 13: Management Model on page 709.

Table 113 Management Class Values

Management Class	Value	Description	Required Support for Class	Reference Section
Subn	0x01	Subnet Management class (LID routed)	All channel adapters, switches, and routers.	14.2 Subnet Management Class on page 794
Subn	0x81	Subnet Management class (Directed route)	All channel adapters, switches, and routers.	14.2 Subnet Management Class on page 794
SubnAdm	0x03	Subnet Administration class	All channel adapters, switches, or routers, hosting a subnet manager	15.2 SA MADs on page 883
Perf	0x04	Performance Management class	All channel adapters, switches, and routers.	16.1 Performance Management on page 930
BM	0x05	Baseboard Management class (tunneling of IB-ML commands through the IBA subnet)	All channel adapters, switches, and routers.	16.2 Baseboard Management on page 973.

Table 113 Management Class Values (Continued)

Management Class	Value	Description	Required Support for Class	Reference Section
DevMgt	0x06	Device Management class	Optional.	16.3 Device Management on page 985
ComMgt	0x07	Communication Management class	All channel adapters that support RC,UC or RD.	16.7 Communication Management on page 1011
SNMP	0x08	SNMP Tunneling class (tunneling of the SNMP protocol through the IBA fabric)	Optional	16.4 SNMP Tunneling on page 998
Vendor	0x09-0x0F 0x30-0x4F	Vendor Specific classes	Optional	16.5 Vendor-specific on page 1005
Application	0x10-0x2F	Application Specific classes. Refer to Annex “Application Specific Identifiers” for a listing of Application Specific Management class values currently assigned.	Optional	16.6 Application-specific on page 1008
	0x00 0x02 0x50-0x80 0x82-0xFF	Reserved		

With respect to the column labeled Required Support for Class, an indication that support is required indicates that at least some aspects of the class must be supported. Complete details of which aspects are mandatory and which aspects are optional are specified in the corresponding reference section.

13.4.5 MANAGEMENT CLASS METHODS

Methods define the operations that a management class supports. In addition to supporting methods common to multiple classes, each management class may define additional class specific methods.

The upper bit of the Method field is designated as the response bit (R). It is used to distinguish three types of messages based upon the type of method included in the header as follows:

- Message methods are methods for which no response is ever generated. The R bit is not set (i.e. it is 0) and the corresponding method with the R bit set is reserved and not used.

- Request methods are methods for which a response may be generated. The R bit is not set (i.e. it is 0) and the corresponding method with the R bit set is defined and potentially used to convey a response.
- Response methods are methods generated in response to receipt of a request method. The R bit is set (i.e. it is 1) and the corresponding method with the R bit not set is defined and used to trigger (request) the response.

See section 13.4.6 Management Messaging on page 723 for required request/response behavior.

C13-6: The method names and method values shown in Table 114 Common Management Methods on page 722 **shall** be used in a manner consistent with the descriptions contained in 13.4.6 Management Messaging on page 723.

C13-7: The values assigned to the common methods **shall** not be used for any class-dependent method even if the common method is not supported.

C13-8: Class specific methods defined to be requests and responses **shall** conform to the request response definitions in this section, the request response requirements specified in Section 13.4.6.4 TransactionID usage on page 731, and **shall** use the R bit according to the semantics of types of methods defined above.

Table 114 Common Management Methods

Name	Type	Value (including R bit)	Description
Get()	Request	0x01	Request (read) an attribute from a channel adapter, switch, or router. See 13.4.6.1.1 Get()/GetResp() on page 724.
Set()	Request	0x02	Request a set (write) of an attribute in a channel adapter, switch, or router. See 13.4.6.1.2 Set()/GetResp() on page 724.
GetResp()	Response	0x81	The response from an attribute Get() or Set() request. See 13.4.6.1.1 Get()/GetResp() on page 724 and 13.4.6.1.2 Set()/GetResp() on page 724.
Send()	Message	0x03	Send a datagram. Does not require a response. See 13.4.6.1.3 Send() on page 726.
Trap()	Message	0x05	An unsolicited datagram sent from a channel adapter, switch, or router indicating an event occurred that may be of interest. See 13.4.6.1.4 Trap() on page 726 and 13.4.9 Traps on page 741.
Report()	Request	0x06	Used to forward an event/trap/notice to interested party. See 13.4.6.1.6 Report()/ReportResp() on page 726 and 13.4.11 Event Forwarding on page 745.

Table 114 Common Management Methods (Continued)

Name	Type	Value (including R bit)	Description
ReportResp()	Response	0x86	Response to a Report(). See 13.4.6.1.6 Report()/ReportResp() on page 726 and 13.4.11 Event Forwarding on page 745.
TrapRepress()	Message	0x07	Instruct a Trap() sender to cease sending a repeated Trap(). See 13.4.6.1.5 TrapRepress() on page 726 and 13.4.9 Traps on page 741 for usage.
		0x00, 0x04, 0x08-0x0F, 0x80, 0x82-0x85, 0x87-0x8F	Reserved.
		0x10-0x7F, 0x90-0xFF	Class-specific methods. Use is defined by the class.

For Get(), Set(), GetResp(), and Send() methods, the combinations of method and attribute that are valid are class specific and are specified in the respective class sections in Chapter 14: on page 794, Chapter 14: on page 794, and Chapter 16: on page 930.

For Trap() and TrapRepress(), Report(), and ReportResp() methods, attribute usage is specified in Sections 13.4.9 Traps on page 741, and 13.4.11 Event Forwarding on page 745.

13.4.6 MANAGEMENT MESSAGING

13.4.6.1 METHODS AND MESSAGE SEQUENCING

Interactions using MADs are in a number of cases organized into request-response pairs such as Get()/GetResp(), Set()/GetResp(), and Report()/ReportResp(). In such cases, the requester may retry requests.

C13-8.1.1: The minimum interval between retries of request messages (all of which use the same TID) **shall** be greater than or equal to the response time specified in C13-15.1.1: on page 730. The requester **shall** cease to retry a given request when it receives a matching response MAD (see C13-19.1.1: on page 731).

Whether a requester retries requests until it receives a matching response is vendor-specific. However, since requests (and all MADs) are unreliable datagrams, there can be no guarantee that the request has been received unless a response is received by the requester; so in the absence of a response, retries may be appropriate. The maximum number of retries performed before concluding that a response will never be received is also vendor-specific.

Responders generate responses as appropriate and required for each request MAD received as specified in sections 13.4.6.1.1 Get()/GetResp() on page 724, 13.4.6.1.2 Set()/GetResp() on page 724, and 13.4.6.1.6 Report()/ReportResp() on page 726

C13-9: Responders **shall** not coalesce responses.

The subsequent ladder diagrams illustrate management request / response behavior for valid MADs. The operations defined below assume the receipt of a valid MAD. A MAD is valid if it satisfies all applicable validation checks as specified in Section 13.5.3 MAD Validation on page 755.

13.4.6.1.1 GET()/GETRESP()

Get() requests the read of an attribute from a channel adapter, switch, or router, as illustrated in Figure 146 on page 724.

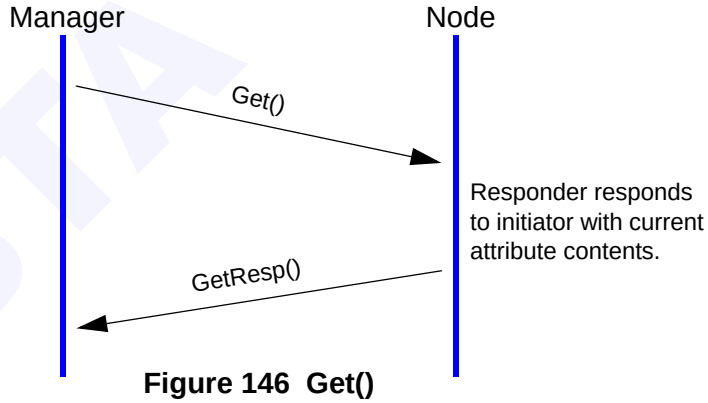


Figure 146 Get()

C13-10: This compliance statement is obsolete and has been replaced by C13-10.1.1:

C13-10.1.1: In response to a valid Get(), the responder **shall** respond with a GetResp() MAD.

The attribute contained in the GetResp() is determined according to the specific MADHeader:MgmtClass and MADHeader:AttributeID in the request

13.4.6.1.2 SET()/GETRESP()

Set() informs the recipient to set values maintained by the recipient according to the values contained in the attribute conveyed in MADHeader:Data, as illustrated in Figure 147 on page 725.

C13-11: This compliance statement is obsolete and has been replaced by C13-11.1.1:.

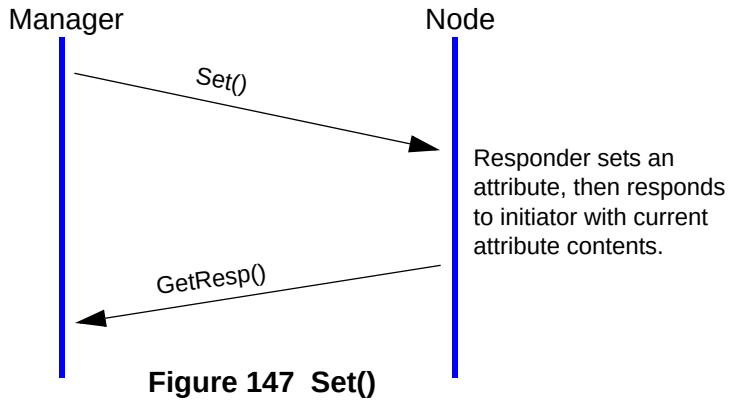


Figure 147 Set()

C13-11.1.1: In response to a valid Set(), the responder **shall** respond with a GetResp() MAD.[20]

C13-12: This compliance statement is obsolete and has been replaced by C13-12.1.1:.

C13-12.1.1: The attribute contained in a GetResp() response to a Set() **shall** be the same as that maintained by the responder when the GetResp() is returned, except as specified in C13-43: on page 754 (redirection) or where otherwise required by the class. Any externally visible effects that result from the Set(), such as attribute component values, **shall** have been made externally visible prior to returning the GetResp() except where otherwise required by the class.

If one or more components of the attribute contained in a Set() have invalid values, the resultant attribute maintained by the recipient is typically implementation-specific, unless otherwise specified by the class. For example, some attribute components may be changed as specified by the Set() and some may not. Nevertheless, the attribute returned in the GetResp() will have the attribute contents maintained by the recipient.

Implementors must be aware that it is possible for retries of a Set() request to result in different GetResp() responses. For example, assume a Set() reached the responder and was successfully processed by the responder, but the corresponding GetResp() was dropped in the fabric. The sender then has no indication that the Set() was processed, and as a result may retry the Set(). Such a retry may result in a GetResp() with the Status field set to an error code because the Set() request was received twice by the responder. This may be the case for a Set () that causes some entity to be deleted from a table; the retried Set() will attempt to delete an entity that no longer exists and as a result returns an error.

20. Exception: A Set() of PortInfo:PortPhysicalState to Disabled does not require that a GetResp() be sent from the port which has been Disabled.

13.4.6.1.3 SEND()

Send() sends data from one entity to another on a class specific basis. If the class specific operations require reliability on top of the unreliable datagram service, higher level protocols may be defined based upon exchanges of send type MADs. Such higher level protocols are class specific and are defined either in the class specific sections or other sections referred to therein.

13.4.6.1.4 TRAP()

Trap() indicates an event occurred at a channel adapter, switch, or router, as illustrated in Figure 148 on page 726. See Section 13.4.9 Traps on page 741 for the specification of Trap() usage and behavior.

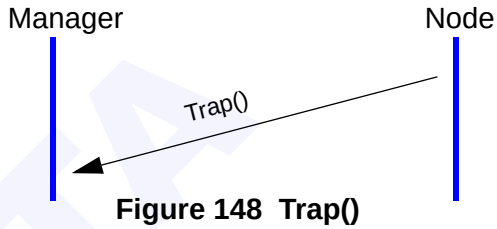


Figure 148 Trap()

13.4.6.1.5 TRAPREPRESS()

TrapRepress() instructs a Trap() sender to cease sending a Trap() it is currently sending. See Section 13.4.9 Traps on page 741 for a complete specification of traps and TrapRepress().

The intended usage of TrapRepress() is shown in Figure 149 on page 726 below.

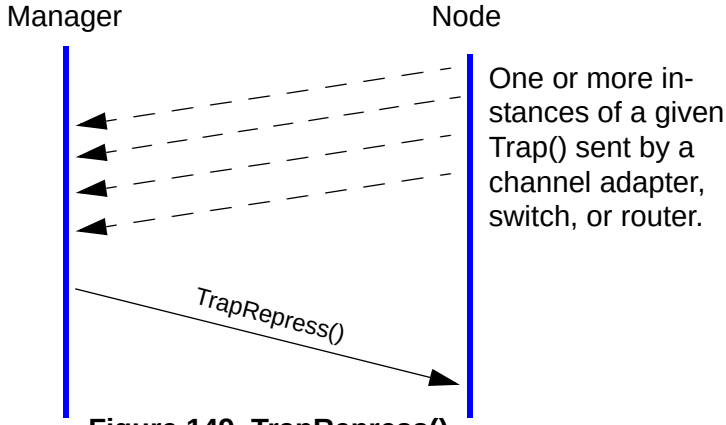


Figure 149 TrapRepress()

13.4.6.1.6 REPORT()/REPORTRESP()

Report() and ReportResp() MADs are used to forward traps directed to a GSM to parties who have subscribed for Trap() forwarding. See 13.4.11 Event Forwarding on page 745 for the complete specification of the event forwarding mechanism.

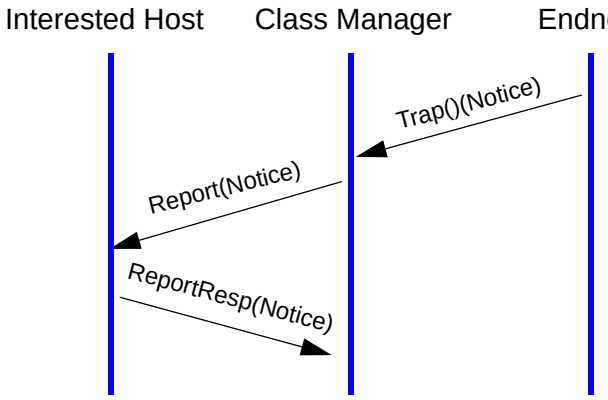


Figure 150 Forwarding Trap()s/Notices from the class manager

C13-12.1.2: The recipient of a Report(Notice) MAD **shall** respond with a ReportResp() MAD with the transaction ID matching that used in the Report() and an empty notice attribute (see 13.4.8.2 Notice on page 737) unless otherwise specified by a compliance statement for that class.

There are issues of ordering concerning when events occur and when Report()s of those events are received. For example: Assume event A is detected by a manager prior to event B. The manager could wait until a subscriber to both A and B has responded (via ReportResp()) to event A before Report()ing event B to that subscriber; or the manager could wait until all subscribers to event A have responded prior to Report()ing event B to any subscriber. Whether any such ordering is maintained is vendor-specific.

13.4.6.2 TIMERS AND TIMEOUTS

A management entity may use the IBA-defined management timeout and response time values to bound the amount of time a requester waits for a response.

Margins for measurement of management time periods are as follows:

- if the period is an upper bound, the margins are -0% and +15%
- if the period is a lower bound, the margins are -15% and +0%
- otherwise the margins are +/- 15%

13.4.6.2.1 PORTINFO:SUBNETTIMEOUT

PortInfo:SubnetTimeout specifies the maximum expected propagation delay, which depends upon the configuration of the switches, to reach any other port in the subnet from the port with which this instance of PortInfo is associated. The duration of time is calculated as

$$4.096\ \mu\text{sec} \times 2^{\text{PortInfo:SubnetTimeout}}$$

requesters may use this value along with the appropriate RespTimeValue (see below), to determine how long to wait for a response to a request before taking other action.

Traps are subject to maximum Trap() rate injection constraints based upon PortInfo:SubnetTimeout. See 13.4.9 Traps on page 741 for the usage of PortInfo:SubnetTimout with respect to traps.

13.4.6.2.2 RESPTIMEVALUE

The IBA defined RespTimeValue specifies the expected maximum time interval between reception of an MAD and transmission of the associated response or between the associated port's transmission of successive MADs that are part of a multiple MAD sequence. Requesters may use this value along with the appropriate SubnetTimeout (above), to determine how long to wait for a response to a request, or, how long to wait for a succeeding MAD in a multi MAD sequence, as described in 13.4.6.3 Timeout/Timer Usage on page 730. The duration of time is calculated as

$$4.096\ \mu\text{sec} \times 2^{\text{RespTimeValue}}$$

where the applicable RespTimeValue is selected according to C13-14: on page 728 below.

C13-13: This compliance statement is obsolete and has been replaced by C13-13.1.1:.

C13-13.1.1: The time period between beginning the reception of a request packet for a Get() of ClassPortInfo or PortInfo on a port and an agent's beginning the transmission of the corresponding response packet **shall** be less than or equal to 4.3 seconds.

Note that 4.3 seconds corresponds to a timer value of 20, since $4.096\ \mu\text{sec} \times 2^{20} = 4.3\ \text{sec}$. The purpose of compliance statement C13-13.1.1 is to provide a value that a requester can use in computing an appropriate timeout when initially acquiring the correct RespTimeValue.

C13-14: The RespTimeValue applicable to a given situation depends upon the operation being performed and the MAD sequences involved. The appropriate RespTimeValue **shall** be determined as follows:

- If MADHeader:MgmtClass is Subn or Directed Route Subn the applicable RespTimeValue is conveyed by PortInfo:RespTimeValue (see 14.2.5.6 PortInfo on page 821 for the definition of PortInfo) of the relevant port as identified below.

- If MADHeader:MgmtClass is any other than Subn or Directed Route Subn, and the MADHeader:Method is not Report(), the applicable RespTimeValue is conveyed by ClassPortInfo:RespTimeValue (see 13.4.8.1 ClassPortInfo on page 734 for the definition of ClassPortInfo) of the relevant port as identified below.
- If the MADHeader:Method is Report(), the applicable RespTimeValue is conveyed by InformInfo:RespTimeValue (see 13.4.8.3 InformInfo on page 739 for the definition of InformInfo) specified by an event subscriber at the time of subscription.

C13-15: In the case of MAD sequences other than Report(), Report-Resp(), the port used to determine the applicable RespTimeValue **shall** be determined as follows:

- For MAD request-response exchanges consisting of a single packet request followed by a single packet response, the applicable RespTimeValue associated with the responding port **shall** indicate the expected maximum interval between receipt of the request at that port and initiation of transmission of the corresponding response.
- For MAD request-response exchanges including a multipacket request sequence followed by a response, the applicable RespTimeValue associated with the sending port **shall** indicate the expected maximum interval between initiation of transmission of successive packets in the multipacket request sequence. The applicable RespTimeValue associated with the receiving port indicates the expected maximum interval between receipt of the last packet of the multipacket request sequence at that port and initiation of transmission of the corresponding response.
- For MAD request-response exhanges including a multipacket response, the applicable RespTimeValue associated with the responding port **shall** indicate the expected maximum interval between receipt of the last packet of the request at that port and initiation of transmission of the response. The applicable RespTimeValue associated with the responding port also indicates the expected maximum interval between the initiation of transmission of successive packets in the multipacket response sequence.
- For MAD request-response exchanges using the windowing protocol defined in 13.6 Reliable Multi-Packet Transaction Protocol on page 770, the applicable RespTimeValue associated with the port originating the request **shall** indicate the expected maximum time within which the requester will request more packets following the last packet in a burst of response packets.
- For operations requiring transmission of a sequence of multiple MADs not classified as requests (e.g. a succession of Send()s conveying fragments of an SNMP frame), the applicable RespTimeValue

associated with the port sending the sequence **shall** indicate the expected maximum interval between initiation of transmission of successive packets in the sequence.

C13-15.1.1: In the case of the MAD sequence Report(), ReportResp(), InformInfo:RespTimeValue specified by an event subscriber at the time of subscription **shall** indicate the expected interval between receipt of the Report() at that port and initiation of transmission of the corresponding ReportResp().

Send(), Trap(), or TrapRepress() do not have an associated response MAD (Send() MADs exchanged as part of a higher level protocol are not request/response sequences in this context). As such, the IBA-defined management timeout and response times are not applicable. Note that while TrapRepress() may be sent as a result of the sending of a Trap(). Trap() and TrapRepress() are classified as messages not as requests or responses and do not constitute a request/response sequence.

13.4.6.3 TIMEOUT/TIMER USAGE

In general, the expected maximum time interval between transmission of a request and receipt of the associated response is

$$4.096\ \mu\text{sec} \times (2^{CommTimeValueOut} + 2^{CommTimeValueIn} + 2^{RespTimeValue})$$

where *RespTimeValue* is determined according to 13.4.6.2.2 RespTimeValue on page 728, and *CommTimeValueOut* and *CommTimeValueIn* are either:

- both equal to PortInfo:SubnetTimeout, if the path is entirely within a single subnet; or
- PathRecord:PacketLifeTime values, with *CommTimeValueOut* using the path from requester to responder, and *CommTimeValueIn* using the path from responder to requester (see 15.2.5.16 PathRecord on page 899).

The use of PathRecord:PacketLifeTime is preferable when a path is known, since it will often be smaller than PortInfo:SubnetTimeout because it refers to a single path, not an upper bound on all paths. PacketLifeTime is, however, not always available; for example, it is not available before first contact has been made with Subnet Administration (see 15.4.1.2 Access Restrictions For Other Attributes on page 922).

C13-16: For request/response sequences, timers **shall** be started for each request transmitted and reset upon arrival of the corresponding response MAD.

C13-17: This compliance statement is obsolete and has been deleted.

13.4.6.4 TRANSACTIONID USAGE

The contents of the TransactionID (TID) field are implementation-dependent.

C13-18: This compliance statement is obsolete and has been replaced by C13-18.1.1:.

C13-18.1.1: When initiating a new operation, MADHeader:TransactionID **shall** be set to such a value that within that MAD the combination of TID, SGID, and MgmtClass is different from that of any other currently executing operation. If the MAD does not have a GRH, its SLID is used in the combination in place of an SGID. Repeated Trap() messages for the same event may be regarded as continuing a 'currently executing' operation as long as the Trap() can be repeated and no corresponding TrapRepress() has been received, or they may be regarded as initiating new operations.

C13-19: This compliance statement is obsolete and has been replaced by C13-19.1.1:.

C13-19.1.1: Note that the above implies that recipients of messages **shall** use the combination of TID, SGID (or SLID), and MgmtClass to uniquely associate messages or message sequences, not just the TID.

C13-20: When constructing a request that consists of sequence of MADs, requesters **shall** set MADHeader:TransactionID in each MAD that is part of the sequence to an identical value.

C13-21: When constructing a response, responders **shall** set MADHeader:TransactionID in the response equal to MADHeader:TransactionID in the corresponding request.

C13-22: Where a response is made up of multiple MADs, MADHeader:TransactionID in each MAD in the response **shall** be set equal to MADHeader:TransactionID in the corresponding request.

C13-23: Where an operation defined for an IBA management class requires a sender to send a succession of MADs of type message to effect the operation (e.g. an SNMP PDU being tunneled through IBA), the sender **shall** set MADHeader:TransactionID in each MAD that is part of the sequence to an identical value.

13.4.7 STATUS FIELD

All MADs contain a status field. The status field is used in MADs of type response to convey information about the disposition of the request or conditions associated with disposition of the request.

The status field consists of 16 bits. The eight low order bits of the field are used for indications common to all classes. The eight high order bits of the field are used for class specific indications. Class specific status indications are defined in the class specific sections of Chapter 14: Subnet Management on page 794, Chapter 15: Subnet Administration on page 882, and Chapter 16: General Services on page 930.

C13-24: For messages of type Response (see 13.4.5 Management Class Methods on page 721), the usage of the low order 8 bits **shall** be set as specified in Table 115 MAD Common Status Field Bit Values on page 732.

Table 115 MAD Common Status Field Bit Values

Bit	Name	Meaning
0	Busy	Temporarily busy. MAD discarded. This is not an error.
1	RedirectionRequired	Redirection. This is not an error.
2-4	Code for invalid field	0 - no invalid fields 1 - Bad version. Either the base version, or the class version, or the combination of the two is not supported. 2 - The method specified is not supported 3 - The method/attribute combination is not supported 4-6: Reserved 7 - One or more fields in the attribute or attribute modifier contain an invalid value. Invalid values include reserved values and values that exceed architecturally defined limits.
5-7	Reserved	
8-15	Class Specific	The use of these bits is class specific.

C13-25: This compliance statement is obsolete and has been deleted.

Except as stated in C13-24: on page 732, the use of the status field is class-specific.

13.4.8 MANAGEMENT CLASS ATTRIBUTES

Attributes define the data which a management class manipulates. Each management class defines its own set of attributes.

Attributes are composite structures consisting of components typically representing hardware registers in channel adapters, switches, or routers. Each attribute is assigned a unique Attribute ID.

Depending upon the attribute, components may be read only, read/write, or reserved.

Some attributes have associated AttributeModifiers (AMs) which further qualify or modify the application of the attribute. The use of the AM is attribute-specific and usage is defined where the attribute is defined.

C13-26: When the AM is not used it **shall** be set to all zeroes.

It is not possible to selectively set a single component within an attribute. A Get() must be performed to obtain the whole attribute, the single component must be modified in the result and a Set() must be performed to write the whole attribute. No atomicity is implied or provided in this sequence of operations.

C13-27: A given attribute **shall** have the same format for the Get(), Set() and GetResp() methods if used with those methods.

There are three attributes which are common across multiple classes. Table 116 Attributes Common to Multiple Classes on page 734 lists each such attribute, its ID, and the classes where it is used. Attribute IDs less than 0x10 identify common attributes or are reserved. Attribute IDs equal to or greater than 0x10 identify attributes whose definitions are class specific. The structure and content of the common attributes is defined in the following subsections.

The overall offset and alignment of the common attributes are defined by the classes which use them. In particular, if a class specifies additional header data following the common MAD header, then when the class uses the common attributes their position and alignment in the MADs of that class may follow that additional header data.

C13-27.1.1: All managers **shall** use the Attribute IDs shown in Table 116 Attributes Common to Multiple Classes on page 734 for the purposes indicated in that table. The content of the ClassPortInfo, Notice, and InformInfo attributes **shall** be as indicated in 13.4.8.1 ClassPortInfo on page 734, 13.4.8.2 Notice on page 737, and 13.4.8.3 InformInfo on page 739 respectively.

The structure and content of class-specific attributes are defined in the respective class specific sections of Chapter 14: Subnet Management on page 794, Chapter 15: Subnet Administration on page 882, and Chapter 16: General Services on page 930.

The following common attributes are defined:

Table 116 Attributes Common to Multiple Classes

Attribute Name	Attribute ID	Attribute-Modifier	Description	Where Used
	0x0000		Reserved	
ClassPortInfo	0x0001	0x00000000	General and port-specific information for a GS management class	The SA class and all supported GS classes on channel adapters, switches, and routers. See 13.4.8.1 ClassPortInfo on page 734.
Notice	0x0002	0x00000000-0xFFFFFFFF	Information regarding the associated Notice (or Trap() in which case the AttributeModifier **shall** be 0)	All classes supporting traps/notices. See 13.4.8.2 Notice on page 737.
InformInfo	0x0003	0x00000000	Event Subscription	All classes having a class manager supporting event subscription. See 13.4.8.3 InformInfo on page 739.
	0x0004-0x000F		Reserved	
	0x0010-0xFFFF		Class-dependent values.	Usage of values in this range is class specific and is specified in the class specific sections of Chapter 14: on page 794, Chapter 15: on page 882, and Chapter 16: on page 930.

13.4.8.1 CLASSPORTINFO

C13-28: Channel adapters, switches, and routers implementing a GS class agent **shall** implement ClassPortInfo according to the definition specified in Table 117 ClassPortInfo on page 735.

C13-29: The ClassPortInfo attribute **shall** be implemented for every GS class agent supported by a node; it **shall** be implemented on every port through which the GS class agent may be accessed.

C13-30: The ClassPortInfo attribute **shall** be implemented for the SA class by any node on which an SA is located; it **shall** be implemented on every port through which the SA class may be accessed.

The presence of ClassPortInfo for a management class confirms the availability of that management class on a particular channel adapter, switch, or router and provides information about the version of MADs supported by the class on that channel adapter, switch, or router. (Note: support for a given class can be determined directly from capability bits in PortInfo for the port in question. See 14.2.5.6 PortInfo on page 821).

The ClassPortInfo attribute also provides port-specific information for class services on a channel adapter, switch, or router. In addition to being available as the object of a Get() method specifying it as the target, ClassPortInfo is also returned as the result of any Get() or Set() if the requester is being redirected as described in 13.5.2 GSI Redirection on page 753.

ClassPortInfo contains information related to general services traps. If sending Trap() messages is supported on a channel adapter, switch, or router, and if Trap() sending is enabled for this port (nonzero TrapLID), ClassPortInfo defines the destination to which traps for the subject GS class applying to this port are to be sent. See 13.4.9 Traps on page 741. Note that this applies only to general services traps. Subnet management traps do not use this mechanism.

For both redirection and traps, ClassPortInfo provides support for cross-subnet communications by including the information necessary to build a properly formed GRH, see 13.4.9 Traps on page 741 and 13.5.2 GSI Redirection on page 753.

Table 117 ClassPortInfo

Component	Access	Length (bits)	Offset (bits)	Description
BaseVersion	RO	8	0	Current supported MAD Base Version. Indicates that this channel adapter, switch, or router supports up to and including this version.
ClassVersion	RO	8	8	Current supported management class version. Requirements for this channel adapter, switch, or router to support previous class versions is class-specific.
CapabilityMask	RO	16	16	Supported capabilities of this management class, bit set to 1 for affirmation of management support. Bit 0 - If 1, the management class agent generates Trap() MADs Bit 1 - If 1, the management class agent implements Get(Notice) and Set(Notice) Bit 2-7: reserved Bit 8-15: class-specific capabilities.
Reserved	RO	27	32	Reserved
RespTimeValue	RO	5	59	See 13.4.6.2 Timers and Timeouts on page 727.
RedirectGID	RO	128	64	The GID a requester **shall** use as the destination GID in the GRH of messages used to access redirected class services. If redirection is not being performed, this **shall** be set to zero.
RedirectTC	RO	8	192	The Traffic Class a requester **shall** use in the GRH of messages used to access redirected class services. For more on the definition and significance of traffic class see 8.2.2.3 Service Levels on page 223 and 8.3.2 Traffic Class (TClass) - 8 bits on page 225
RedirectSL	RO	4	200	The SL a requester **shall** use to access the class services.

Table 117 ClassPortInfo (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
RedirectFL	RO	20	204	The Flow Label a requester **shall** use in the GRH of messages used to access redirected class services.
RedirectLID	RO	16	224	If this value is non-zero, it is the DLID a requester **shall** use to access the class services. If this value is zero, the redirect requires the requester to use the supplied RedirectGID to request further path resolution from subnet administration. The RedirectGID, the RedirectQP and RedirectP_Key from this redirect response are all valid, but the RedirectSL, RedirectFL, RedirectTC, and RedirectLID will in general not be valid; they must be replaced using a PathRecord obtained from the SA. See the comment about redirection following this table.
RedirectP_Key	RO	16	240	The P_Key a requester **shall** use to access the class services.
Reserved	RO	8	256	Reserved
RedirectQP	RO	24	264	The QP that a requester **shall** use to access the class services.
RedirectQ_Key	RO	32	288	The Q_Key associated with the RedirectQP. This Q_Key **shall** be set to the well known Q_Key.
TrapGID	RW	128	320	The GID to be used as the destination GID in the GRH of Trap() messages originated by this service. If all zeroes, no GRH is inserted in Trap() messages.
TrapTC	RW	8	448	The Traffic Class to be placed in the GRH of Trap() messages originated by this service. For more on the definition and significance of traffic class see 8.2.2.3 Service Levels on page 223 and 8.3.2 Traffic Class (TClass) - 8 bits on page 225.
TrapSL	RW	4	456	The SL that **shall** be used when sending Trap() messages originated by this service.
TrapFL	RW	20	460	The Flow Label to be placed in the GRH of Trap() messages originated by this service.
TrapLID	RW	16	480	The DLID to where Trap() messages **shall** be sent by this service. If all zeroes, traps **shall** not be sent from this port.
TrapP_Key	RW	16	496	The P_Key to be placed in the header for traps originated by this service.
TrapHL	RW	8	512	The Hop Limit to be placed in the GRH of Trap() messages originated by this service. This specifies the maximum number of routers through which the message containing the GRH specified here may pass. The default value is 255.
TrapQP	RW	24	520	The QP to which Trap() messages originated by this service **shall** be sent.
TrapQ_Key	RW	32	544	The Q_Key associated with the TrapQP. This Q_Key **shall** have the high order bit set. See 10.2.5 Q_Keys on page 439 for a description of the significance of setting the high order bit.

Comment About Redirection: Redirection may be fairly complex on certain fabric topologies. Simple InfiniBand CAs may not be able to fully resolve a path to another port it may supply, while complex InfiniBand management applications may do this as a matter of course. In the former case the simple CA can send a redirect to a requester that sets the RedirectLID component in the ClassPortInfo Attribute to zero. A zero value RedirectLID indicates that the requester must request a PathRecord from the SA using the supplied RedirectGID and requester's own source information. Refer to 15.2.5.16 PathRecord on page 899.

13.4.8.2 NOTICE

The Notice attribute describes an exception or other channel adapter, switch, or router event. It is used by both the Trap() mechanism described in 13.4.9 Traps on page 741 and the Notice mechanism described in 13.4.10 Notice Queue on page 743.

C13-30.1.1: If a management class defines any Notice attributes, then the class manager **shall** support both the reception of Trap()s and the Notice Queue operations of polling and deleting entries, unless otherwise specified by a compliance statement for that class.

C13-30.1.2: An agent **shall** either send a Trap() or store a notice in its Notice Queue or both when the condition or event corresponding to the notice occurs, unless otherwise specified by a compliance statement for that class.

o13-1: This compliance statement is obsolete and has been replaced by o13-1.1.1:.

o13-1.1.1: Channel adapters, switches, and routers implementing Notice attributes **shall** conform to the definition specified in Table 118 Notice on page 737.

Table 118 Notice

Component	Access	Length (bits)	Offset (bits)	Description
IsGeneric	RO	1	0	If set to 1, notice is generic, else is vendor specific
Type	RO	7	1	Enumeration indicating type of Trap()/notice: 0 - Fatal 1 - Urgent 2 - Security 3 - Subnet Management 4 - Informational 5-0x7E - Reserved 0x7F - Empty notice. All other fields are meaningless.

Table 118 Notice (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
ProducerType / VendorID	RO	24	8	If generic, indicates the type of the event's producer: 1 - Channel Adapter 2 - Switch 3 - Router 4 - Class Manager 0, 5-0xFFFFFF - Reserved If not generic, indicates the 24 bit IEEE OUI[a] assigned to the vendor.
TrapNumber / DeviceID	RO	16	32	If generic, indicates a class-defined trap number. Number 0xFFFF is reserved. If not generic, this is Device ID information as assigned by device manufacturer.
IssuerLID	RO	16	48	In a GetResp(Notice) message: the base LID of the port from which this Notice is retrieved. In a Trap() message: the base LID of the port that emitted the Trap(). In a Report() message: • If the Report() message resulted from a Trap() message or from polling a Notice Queue, the IssuerLID used in that case. • If the Report() message was fabricated by the class manager on behalf of a particular node, this may be the a LID of the subject node, as specified by the management class. • Otherwise, the base LID of the port emitting the Report() message (e.g., SM traps 64 and 65; see 14.2.5.1 "Notices and Traps" on page 812.
NoticeToggle	RW	1	64	For Notices retrieved from Notice Queues, alternates between zero and one after each Notice is cleared. See 13.4.10 Notice Queue on page 743. For Trap()s, this shall be set to 0. In a Report() message, this value is undefined.
NoticeCount	RW	15	65	For Notices retrieved from Notice Queues, indicates the number of notices queued on this channel adapter, switch, or router. See 13.4.10 Notice Queue on page 743. For Traps, this shall be set to 0. In a Report() message, this value is undefined.
DataDetails	RO	432	80	If generic, data details is disambiguated by management class and TrapNumber. Otherwise disambiguation is vendor defined.

Table 118 Notice (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
IssuerGID	RO	128	512	This field **shall** only be present in Notice attributes sent in GMPs using the Report() method. It **shall** not be present in any other uses of the Notice attribute, such as those sent using the Trap() method and those sent using Set(Notice), Get(Notice), or GetResp(Notice) operations targeting a Notice Queue (see 13.4.10 Notice Queue on page 743)[b]. If IssuerGID is present, its value identifies the port that was the source of the event, i.e.: • If the Report() is sent as a result of a Trap() or GetResp(Notice) that contained a GRH, then IssuerGID **shall** be a GID of the port identified by the SGID field in that GRH. • Otherwise, IssuerGID **shall** be set to a GID of the port identified by IssuerLID.

a. An OUI is a 24 bit globally unique assigned number referenced by various standards. OUI is used in the family of 802 LAN standards, e.g., Ethernet, Token Ring, etc. See http://standards.ieee.org.
b. Trap()s, as well as Set()s and Get()s on Notice Queues, may be sent using SMPs. However, there is insufficient room for IssuerGID in Notice attributes carried by SMPs because of SMPs' provision for directed route information. There is sufficient room when Notice attributes are carried by GMPs.

Certain operations involving the Notice attribute require the use of an empty notice. See 13.4.10 Notice Queue on page 743 and 13.4.11 Event Forwarding on page 745. An empty notice is a Notice attribute in which the type field is set to 0x7F. The only valid field in an empty notice is the type field, the contents of all others should be considered meaningless.

13.4.8.3 INFORMINFO

The InformInfo attribute provides information for subscribing to a class manager for event forwarding. See 13.4.11 Event Forwarding on page 745.

o13-2: This compliance statement is obsolete and has been replaced by o13-2.1.1:.

o13-2.1.1: Channel adapters, switches, and routers implementing the InformInfo attribute **shall** conform to the definition specified in Table 119 InformInfo on page 739.

Table 119 InformInfo

Component	Type	Length (bits)	Offset (bits)	Description
GID	RW	128	0	Specifies specific GID to subscribe for. Set to all zeroes if not desired. See 13.4.11 Event Forwarding on page 745.

Table 119 InformInfo (Continued)

Component	Type	Length (bits)	Offset (bits)	Description
LIDRangeBegin	RW	16	128	Ignored if GID is nonzero. Specifies the lowest LID in a range of LID addresses to subscribe for. Address 0xFFFF denotes all endports managed by the manager to which this InformInfo is directed.
LIDRangeEnd	RW	16	144	Ignored if GID is nonzero. Specifies the highest LID in a range of LID addresses to subscribe for. Set to 0 if no range desired. Ignored if LIDRangeBegin is 0xFFFF.
Reserved	RO	16	160	Reserved
IsGeneric	RW	8	176	If set to 1, forward generic traps or notices. If set to 0, forward all vendor specific traps or notices. Values above 1 are undefined.
Subscribe	RW	8	184	If set to 1, subscribe If set to 0, unsubscribe. Values above 1 are undefined.
Type	RW	16	192	Enumeration indicating the type of the Trap() or notice. Valid values are: 0 - Fatal 1 - Urgent 2 - Security 3 - Subnet Management 4 - Informational 0xFFFF - forward all
TrapNumber /DeviceID	RW	16	208	If not generic, this is device ID information as assigned by device manufacturer. If generic, indicates trap number. Number 0xFFFF means forward any TrapNumber/DeviceID.
QPN	RW	24	224	Ignored except when subscribe=0 (an unsubscribe request). Queue pair to which Report()s were sent as a result of a corresponding subscription. If no subscription for this Report() with this QPN exists, the request to unsubscribe performs no action and produces GetResp() with status indicating an invalid field value (see Table 115 MAD Common Status Field Bit Values on page 732).
Reserved	RO	3	248	Reserved
RespTimeValue	RO	5	251	See 13.4.6.2.2 RespTimeValue on page 728.
Reserved	RO	8	256	Reserved

Table 119 InformInfo (Continued)

Component	Type	Length (bits)	Offset (bits)	Description
ProducerType/VendorID	RW	24	264	If generic, indicates the type of the event's producer: 1 - Channel Adapter 2 - Switch 3 - Routero 4 - Class Manager 0,5-0xFFFFFE - Reserved 0xFFFFFF - Indicates that all events produced or received by this manager should be forwarded. Implementing this behavior is optional; if not implemented, the usual response to invalid field values is returned (see Table 115, "MAD Common Status Field Bit Values," on page 732)." If not generic, indicates the 24 bit IEEE OUI assigned to the vendor.

Nodes can unsubscribe from any subscriptions they have made by setting InformInfo:Subscribe to 0 and providing the relevant components of the InformInfo attribute. It is the node's responsibility to clean up any stale subscription on application failure. For example, if an application exits, the supervisor could send unsubscribe messages to all the managers that support event forwarding in its partitions. These messages would unsubscribe all the UD QPs that have the (privileged) reserved management Q_Key and were used by the application.

13.4.9 TRAPS

Traps are asynchronous notifications for the purpose of alerting an entity within another channel adapter, switch, or router about exception conditions or other events of interest at a given channel adapter, switch, or router within the subnet.

C13-31: This compliance statement is obsolete and has been replaced by C13-30.1.2: on page 737.

The conditions and events corresponding to notices are defined by management classes, which also define whether these are optional or required for that class.

See 13.4.10 Notice Queue on page 743 for a description of Notice support.

For the subnet management class, traps originate at a CA, switch, or router and are always sent to the master subnet manager managing the originator. The managing SM is always on the same subnet since cross subnet communications is not allowed for subnet management class MADs. Subnet management traps are always allowed and there is no direct mechanism for preventing a CA, switch, or router from sending traps

(note, there may be actions which as a side effect cause cessation of traps, i.e. downing a port, but these are not considered direct mechanisms).

For each GS management class, whether traps are allowed to be sent for that class is specified in ClassPortInfo for that class.

C13-32: If ClassPortInfo:TrapLID for a particular port and class is zero, traps **shall** not be generated from that port for that class.

o13-2.a1: If Trap() generation is supported, the destination address and certain other necessary message parameters **shall** be obtained from ClassPortInfo as indicated in Table 117 ClassPortInfo on page 735 and the source LID **shall** be set to the PortInfo:LID of the originating port.

The destination for traps may be in the same subnet or in another subnet. If traps are to be delivered to a destination not in the same subnet, the ClassPortInfo:TrapGID is non zero and a GRH is required in the Trap() message. If ClassPortInfo:TrapGID is zero, the Trap() destination is within the same subnet and no GRH is included in the message.

If a GRH is required, the source GID in the GRH is the GIDIndex0 of the originating port and other fields in the GRH are either fixed or are given values drawn from counterpart fields in ClassPortInfo. 8.3 Global Route Header on page 225 specifies the requirements applicable to GRHs.

Traps are always sent to the destination identified in ClassPortInfo and only to that destination. It is the responsibility of any entity that sets ClassPortInfo fields to assure that the values programmed are consistent. That is, the combination of GID/DLID, P_Key, port, etc. must be consistent with addressing of and access rules applicable to the specified port.

Traps may be issued by any channel adapter, switch, or router on the subnet. Channel adapters, switches, and routers may repeat sending of a Trap()

o13-3: Channel Adapters, switches and routers **shall not** send traps for any given management class at a rate greater than the Trap() rate limit specified. For a given port, the Trap() rate limit **shall** be defined as the reciprocal of the time duration determined from PortInfo:SubnetTimeout for that port. See 13.4.6.2.1 PortInfo:SubnetTimeout on page 727.

o13-4: Traps **shall** contain the Notice attribute to identify the Trap(). The Notice attribute is described in 13.4.8.2 Notice on page 737.

o13-5: Trap() originators **shall** use the same MADHeader:TransactionID value for all instances of repeated traps.

Recipients of traps may send TrapRepress() MADs to Trap() originators.

o13-5.1.1: A TrapRepress() MAD **shall** be constructed in accordance with 13.5.4 Response Generation and Reversible Paths on page 768 as if it were a response to the Trap() being repressed, except that the Q_Key shall be set to the Q_Key of the corresponding Trap().

o13-6: Upon receipt of a valid TrapRepress() MAD, the Trap() originator **shall** cease sending the Trap() which matches the Trap() identified by the TrapRepress() MAD. A Trap() being repeatedly sent matches a Trap() identified in a TrapRepress() MAD when both MADHeader:TransactionID in the Trap() MAD matches MADHeader:TransactionID in the TrapRepress() MAD and the Notice attribute in the Trap() MAD matches the Notice attribute in the TrapRepress().

o13-7: If a TrapRepress() is received and no matching Trap() is being sent, the TrapRepress() **shall** be silently dropped and no other action taken.

Sending traps is optional, unless specified otherwise by a compliance statement.

Management classes supplement Trap() handling through the use of the event forwarding mechanism described in 13.4.11 Event Forwarding on page 745.

Although Traps and the Notice Queue (NQ) mechanism (see 13.4.10 Notice Queue on page 743) use the same Notice attribute to describe events or conditions, the Trap() mechanism and the notice queues mechanism are completely independent. There is no requirement that a Notice queue entry be generated when a Trap() is sent or vice versa.

13.4.10 NOTICE QUEUE

The NQ is a repository for storing Notice attributes (see 13.4.8.2 Notice on page 737) associated with the occurrence of an event or the detection of a condition at a channel adapter, switch, or router. Notices in the NQ are queried or deleted using Get(Notice) and Set(Notice) methods respectively.

o13-8: The Notice Queue **shall** operate as a first in first out queue. For Get(Notice), the AM **shall** be 0. This selects the oldest notice attribute saved, that is, the Notice attribute on the top (or front) of the queue.

o13-9: Notice:NoticeCount in a returned Notice attribute **shall** always indicate the number of notices currently on the queue. Performing a Get(Notice) does not remove a Notice from the NQ. Notice:NoticeCount includes the notice returned in response to the Get().

o13-10: This compliance statement is obsolete and has been replaced by o13-10.1.1:

o13-10.1.1: If the queue is empty, the Notice attribute returned **shall** be an empty Notice and Notice:NoticeCount **shall** contain 0. Otherwise the recipient of a Get(Notice) **shall** return a copy of the oldest notice in the Notice attribute in the response.

To clear notices from the head of the Notice Queue, the requester sends a Set(Notice) MAD containing a Notice attribute with:

- Notice:NoticeToggle set to match Notice:NoticeToggle in the channel adapter's, switch's, or router's Notice attribute.
- Notice:NoticeCount set to the number of notices to delete.
- MADHeader:AM = 0

o13-11: Upon receipt of a Set(Notice) MAD, if the recipient implements an NQ, the recipient **shall** perform the following actions:

- If the NoticeToggle value in the Set(Notice) does not match the NoticeToggle value in the Notice attribute on the channel adapter, switch, or router, the Set(Notice) is silently discarded and no other action is taken.
- The oldest notice and successively newer notices up to a total number of notices indicated by Notice:NoticeCount in the Notice attribute contained in the Set(Notice) **shall** be deleted. If Notice:NoticeCount in the request Notice is greater than the number of notices on the queue, the queue is emptied.
- The response to the next Get(Notice) request **shall** return a Notice attribute that corresponds to the new top of the queue and Notice:NoticeCount in the response **shall** reflect the updated count of notices on the queue.
- Since the Notice Queue acts as a FIFO, the only valid value for MADHeader:AM is 0.

The attribute content is undefined in the GetResp() issued in response to a Set(Notice).

The types and number of notices captured by a channel adapter, switch, or router is implementation-dependent. The actual size of the Notice Queue is implementation specific and is not specified by the architecture. Behavior of a full Notice Queue when the channel adapter, switch, or router has another notice to queue is undefined.

Channel adapters, switches, and routers are not required to support the NQ mechanism.

13.4.11 EVENT FORWARDING

Entities can request that events be forwarded to them by subscribing for them. These events are instantiated by traps or notice queue entries. Traps are sent to the class manager, while notice queues are polled by the class manager.

Entities request notification via the event-forwarding subscription mechanism. To subscribe, an interested entity sends a Set(InformInfo) request to the class manager identifying the set of devices for which events are to be forwarded. The set of devices can be identified by the GID or LID of a specific port or a range of LIDs which encompasses many ports. The class manager responds with a GetResp(InformInfo) message to confirm or deny such forwarding.

It is optional for a class manager to implement event forwarding. However, see C13-32.1.1:.

C13-32.1.1: A manager for a class that defines either notices or traps **shall** support event forwarding (see compliance statements o13-12.1.1: on page 745 through o13-17.1.1: on page 747). The SA **shall** perform this function for the SM.

o13-12: This compliance statement is obsolete and has been replaced by o13-12.1.1:.

o13-12.1.1: A manager that supports event forwarding **shall** confirm a valid request for event subscription (see o13-14.1.1: on page 746) by responding with an InformInfo attribute that is a copy of the data in the Set(InformInfo) request.

C13-32.2.1: If a manager receives a duplicate Set(InformInfo) with InformInfo:Subscribe set to 1, one which has all the data in the attribute and the source QPn identical to a prior successful Set(InformInfo), the duplicate Set(InformInfo) **shall** be ignored except for returning the normal non-error response to a successful Set(InformInfo).

o13-13: This compliance statement is obsolete and has been replaced by o13-13.1.1:.

o13-13.1.1: This compliance statement is obsolete and has been deleted.

Requesters wishing to subscribe to event forwarding may determine which managers exist and their locations on the fabric by querying the SA. See 15.4 Operations on page 921 for a discussion of subnet administration including restrictions on access to and use of the SA.

The exchange of MADs to effect subscription to event forwarding is depicted in Figure 151 Subscribing and unsubscribing for forwarding on page 746 below.

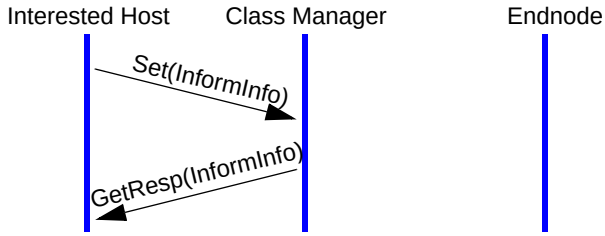


Figure 151 Subscribing and unsubscribing for forwarding

o13-14: This compliance statement is obsolete and has been replaced by o13-14.1.1:.

o13-14.1.1: Except for Set(InformInfo) requests with InformInfo:LIDRangeBegin=0xFFFF, managers that support event forwarding **shall**, upon receiving a Set(InformInfo), verify that the requester originating the Set(InformInfo) and a Trap() source identified by InformInfo:GID or InformInfo:LIDRangeBegin and InformInfo:LIDRangeEnd are permitted to access each other according to the current partitioning. The manager **shall** perform verification by verifying that a valid path exists between the requester and the Trap() source.

This verification can be accomplished by requesting a path between the requestor and a trap source using a SA query operation. If such a path exists, the Set(InformInfo) succeeds. If such a path does not exist, the GID or LID is invalid as a Trap() source. A GID or LID specifies an invalid Trap() source if the associated port is not accessible to the requester under current partitioning.

C13-32.2.2: Managers that support event forwarding **shall**, upon receiving a Set(InformInfo), verify that the requester is allowed to receive all the traps that it subscribes for. If a single Set(InformInfo) is used to subscribe to multiple traps, and the subscriber is not allowed to receive one or more of those traps, the entire Set(InformInfo) **shall** fail and no subscriptions shall be recorded.

o13-15: This compliance statement is obsolete and has been replaced by 13-15.2.1.

o13-15.2.1: If partition or other verification fails on Set(InformInfo), the manager receiving the request **shall** indicate in the response that the operation failed with an invalid attribute status value as defined in 13.4.7

Status Field on page 731 and Table 115 MAD Common Status Field Bit Values on page 732).

o13-16: This optional compliance statement is obsolete and has been deleted.

o13-17: This optional compliance statement is obsolete and has been replaced by o13-17.1.1:.

o13-17.1.1: Managers that support event forwarding and have confirmed a request for event subscription **shall** forward corresponding events to the subscriber using a Report(Notice) MAD, as long as the subscriber and Trap() source are permitted to access each other according to current partitioning.

o13-17.2.1: Managers that support event forwarding and have confirmed a request for event subscription, yet on sending a Report(Notice) to the subscriber receive no ReportResp(Notice) after a vendor-chosen number of retries, or receive an invalid ReportResp(Notice), **shall** permanently discontinue all event forwarding caused by the Set(InformInfo) which created a subscription to that trap source.

o13-17.1.2: If a Set(InformInfo) specified a valid trap source at the time of subscription (see o13-14.1.1: on page 746), yet Trap() forwarding fails because the subscriber and trap source are no longer permitted to access each other according to current partitioning (see o13-17.1.1: on page 747), then the manager **shall** permanently discontinue all event forwarding caused by the Set(InformInfo) which created a subscription to that trap source, except if InformInfo:LIDRangeBegin was 0xFFFF; in the latter case, event forwarding is discontinued only for the now-invalid trap source.

Note that for the case where InformInfo:LIDRangeBegin is 0xFFFF, event forwarding from an invalid trap source may be subsequently resumed if partitioning changes such that it becomes possible for the subscriber and trap source to access each other.

Note also that “permanently discontinue all event forwarding” is meant to indicate that the subscription for forwarding is dropped by the manager; if the source later becomes reachable again by the subscriber, a new Set(InformInfo) is required to re-establish event forwarding, if that is what is desired. (This may not be desired; when the source becomes reachable again, it may have acquired new characteristics, such as new, different software functions, that make such forwarding inappropriate.)

C13-32.1.2: When forwarding Notices to an event subscriber, a manager **shall** construct the Report(Notice) MAD using addressing information

from the original Set(InformInfo) subscription operation as specified in Table 120 Setting Report(Notice) MAD Fields on page 748.

Table 120 Setting Report(Notice) MAD Fields

Components of a Report(Notice) used to forward an event	What that Report(Notice) component is set to: component values from the Set(InformInfo) that created the subscription to that event
LRH:DLID	LRH:SLID
LRH:SL	LRH:SL
BTH:P_Key	BTH:P_Key
BTH:DestinationQP	DETH:SourceQP
DETH:Q_Key	DETH:Q_Key
GRH:DGID[a]	GRH:SGID
GRH:FlowLabel[a]	GRH:FlowLabel
GRH:TClass[a]	GRH:TClass
GRH:HopLmt[a]	GRH:HopLmt

a. Present only if a GRH was present in Set(InformInfo).

Figure 152 Forwarding Trap()s/Notices from the Class Manager on page 748, depicts the MAD exchange associated with forwarding traps to a subscriber. The ReportResp() provides means for the class manager to assure that subscribers receive reports assuming communications with the subscriber is not failed.

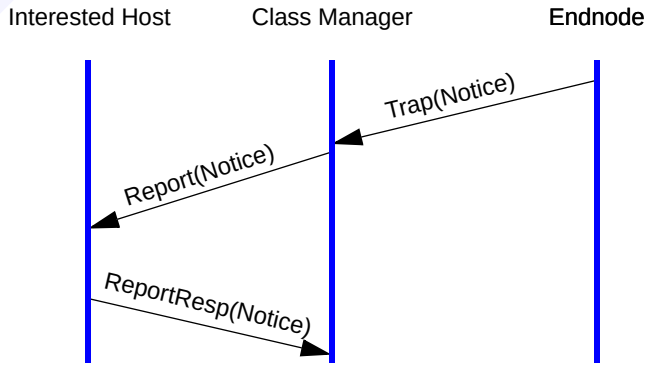


Figure 152 Forwarding Trap()s/Notices from the Class Manager

This forwarding service is available for all management classes. For the Subnet Management Class, event forwarding is not handled by the SM; it is handled indirectly through the SA, as detailed in 15.4.3 Event Forwarding Subsystem on page 923.

13.5 MAD PROCESSING

Non-redirected MADs are distinguished from other packets by the destination queue pair specified in the packet. Two specific queue pair numbers are dedicated to supporting non redirected management operations. Each of the dedicated queue pairs represents a unique interface to one or more management services. These interfaces and the behaviors related to associated services are specified in subsequent sections.

If redirection is in effect, redirected MADs may be directed to a queue pair different from either of the dedicated queue pairs. How a management service is associated with such a queue pair is implementation specific. Such MADs are standard packets and MADs arriving at a port are directed according to the standard procedures for directing packets to queue pairs.

13.5.1 MAD INTERFACES

Figure 153 MAD Interface on page 750, below, depicts the general relationships among management entities and their interfaces to the wire. Note, the figure itself is meant to be representative of a basic channel adapter, switch, or router and is not meant to imply a specific implementation or to imply specific requirements or limitations.

As can be seen in Figure 153 on page 750, QP0 is used for communication with the Subnet Manager (SM) and Subnet Management Agent (SMA); and QP1 is used for communication with General Services Agents.

Note that it is not required by IBA that GS managers use QP1 as the source QP used to send management packets to GS agents. GS managers may send packets from any QP other than QP0. QP1's primary purpose is to be a known QP target to which GS managers can send packets to initially contact a GS agent on a node. Redirection to another non-special QP can be used following that initial communication on QP1, if the agent does this. Such redirection may well be desirable on both functional and performance grounds; for example, the special P_Key matching on QP1 (see 9.6.1 Validating Header Fields on page 272) may in many implementations impact the performance of packet reception on QP1.

Packets arriving at QP0 may be intended for either the SM, if one is present; or for the SMA. Similarly, packets arriving at QP1 may be intended for any of a number of management agents. Any implementation must therefore provide a dispatching function that routes packets on QP0 and QP1 to their appropriate destinations. Similarly, any implementation must also provide other low-level functions on packets received on or destined for those QPs (e.g., directed route processing for QP0). For convenience of description, all such functions are referred to in the specification as being carried out by the Subnet Management Interface (SMI) for QP0 and the General Services Interface (GSI) for QP1. Neither the SMI nor the

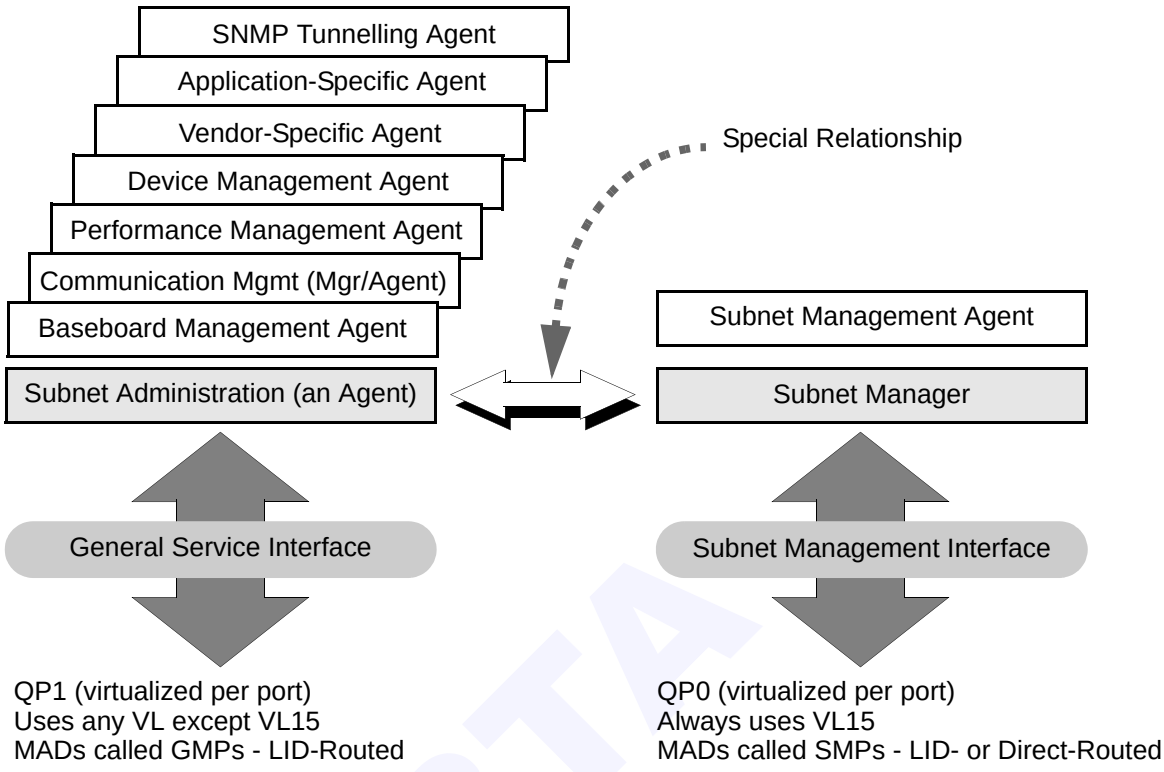


Figure 153 MAD Interface

GSI are architected elements required by any IBA implementation. Any operations stated in compliance statements as being performed “by” the SMI or GSI may be performed in any way that is consistent with the external effects of those compliance statements, specifically those effects which are detectable on communication links or on verb interfaces.

C13-33: For each endport, Subnet Management MADs to be processed at that port **shall** be destined to Queue Pair 0.

C13-34: For each endport, unless redirected (see 13.5.2 GSI Redirection on page 753), SA or GS MADs to be processed by agents at that port **shall** be destined to Queue Pair 1.

Queue pairs 0 and 1 have unique semantics with respect to processing of messages specifying one of them as the destination queue pair. Implementations of QP0 and 1 are not required to follow the semantics associated with other queue pairs with respect to requirements such as posting

and consumption of WQEs, manipulation of an associated completion queue, and so on. Messages arriving at QP0 or QP1 are processed in accordance with the requirements set forth in this section and following Sections: 13.5.3.1 MAD Validation for Subnet Management MADs on page 755, 13.5.3.2.1 MAD Validation at the GSI on page 757, and 13.5.3.2.2 MAD Validation at the SA and GSAs on page 758.

One way for the GSI of CAs supporting QPs other than QP0 and QP1 to perform this disambiguation is to require that all managers send and receive GMPs on QPs other than QP1. This disambiguates manager vs. agent GMPs by reserving QP1 for GMPs sent to agents. Whether this is done is vendor-dependent; it is not required by the InfiniBand architecture. It is, however, allowed. As a result, portability of manager implementations will be enhanced if their implementations do not require use of QP1 for sending GMPs.

13.5.1.1 PROCESSING SUBNET MANAGEMENT PACKETS (SMPS)

The Subnet Management Interface (SMI) is associated with QP0. QP0 is used exclusively for sending and receiving subnet management MADs. Communications with the SMA in a channel adapter, switch, or router is always through the SMI. If a channel adapter, switch, or router hosts a SM, then communications between that SM and the SMA of each channel adapter, switch, or router in the subnet is also through the SMI. Only SMAs and SM communicate through this interface. No other entities may do so.

The MADs of subnet management class are called SMPs.

C13-35: SMPs **shall** not travel beyond the boundaries of a subnet (i.e. through a router).

MADs with a destination queue pair of 0 are validated according to the rules specified in 13.5.3.1 MAD Validation for Subnet Management MADs on page 755.

Validated MADs arriving for QP0 are handled by the SMI. It is not specified how the SMI dispatches the SMPs between the SMA and a possible SM.

C13-36: On an HCA, SMPs not dispatched to the SMA **shall** be posted to the QP0 queue pair exposed above the verb layer.

C13-37: For SMPs dispatched to the SMA, a vendor **shall**

- either never post such SMPs,
- or, always post such SMPs,
- or, offer a vendor specific option to select whether such SMPs are never posted or are always posted,

where posting is with respect to the QP0 queue pair exposed above the verb layer.

13.5.1.2 PROCESSING GENERAL SERVICES MANAGEMENT PACKETS (GMPS)

The General Services Interface (GSI) is associated with QP1. QP1 is reserved exclusively for subnet administration and general services MADs. Unless redirected, GSAs send and receive MADs by means of the GSI. For a description of redirection see 13.5.2 on page 753. Depending upon implementation, the GSI may also provide the interface through which a class manager communicates with corresponding (class specific) GSAs throughout the fabric.

The MADs defined for subnet administration and general services are referred to as GMPs. The GSI acts as a demultiplexor for GMPs, distributing messages destined for QP1 to the appropriate service agent or class manager, based upon MADHeader:MgmtClass in the MAD header. MADs with a destination queue pair of 1 are validated according to the rules specified in 13.5.3.2.1 MAD Validation at the GSI on page 757.

In those cases where the GSI provides an interface for both a class service agent and the corresponding class manager, the determination of the appropriate destination above the GSI demultiplexing is implementation dependent.

Also note that GMPs are unicast in the current specification release. Future specification revisions may extend GMPs to multicast.

On an HCA, the GSI is only aware of agents residing below the verb layer.

C13-38: GMPs dispatched to agents implemented below the verb layer **shall** not be visible above the verb layer.

C13-39: This compliance statement is obsolete and has been replaced by C13-39.1.1:.

C13-39.1.1: GMPs that are not dispatched to agents implemented below the verb layer **shall** be visible above the verb layer. Their reception **shall** consume, in a manner visible by verbs, Receive Work Requests previously posted to the QP on which they were received.

The SL used by a GMP is neither specified nor constrained by virtue of the fact it is a GMP. The choice of SL is outside of the scope of these sections. Note that unlike SMPs which follow special and unique VL rules, GMPs are standard unreliable datagrams subject to and only to the SL/VL usage rules applicable to all unreliable datagrams.

If redirection has been configured for a management class, GMPs destined to the QP specified in the redirection are treated exactly the same as any other unreliable datagram. Since the destination QP is not QP1, they do not appear at the GSI but are delivered directly to the QP specified in the redirection by the IB transport in the same manner as any other unreliable datagram.

C13-40: GSAs that are accessed using redirection **shall** validate arriving MADs according to the same rules as apply for queue pair 1.

Any management entity using a QP that has the well-known Q_Key (0x8001_0000) can assume all Unreliable Datagrams received on that QP are GMPs. This is a consequence of the fact that the well-known Q_Key is reserved for MADs, which is implied by the combination of C9-48: on page 277, C9-49: on page 278, C13-51.1.1: on page 770, and C13-52.1.1: on page 770.

GMPs may contain a GRH and may be forwarded across subnet boundaries. Whether or not a given class manager supports cross subnet communications with corresponding class service agents is implementation dependent.

Table 121 Management Interfaces Summary on page 753 summarizes the properties associated with the above described management interfaces.

Table 121 Management Interfaces Summary

	Subnet Management Interface	General Services Interface
Queue Pair	QP0	QP1 (destination before redirection)
VL	VL 15	not VL15
Partitioning	not enforced	enforced
Q_Key	not enforced	enforced (Q_Key = 0x8001_0000)
Scope	Within subnet only	Routable across subnets
Class Key	Management Key (M_Key)	class dependent

13.5.2 GSI REDIRECTION

By default, the interface from the wire to class service agents is the GSI. A mechanism is provided by which the interface to a given class service agent may be relocated to another queue pair. This mechanism is called redirection and is specified in detail below. The SA as well as each GSA may individually support this mechanism or not. The ClassPortInfo attribute is used to indicate if redirection is supported, and, if so, contains redirection information for MADs of the subject class.

C13-41: If, for a class, redirection is not being used, any GMP destined to the associated class agent via QP1 **shall** be processed by that agent.

C13-42: For any request sent to QP1 with MADHeader:MgmtClass equal to the class value of a class being redirected, a response **shall** be returned containing ClassPortInfo for the class specified in the request.

C13-42.1.1: The response described in C13-42: **shall** also occur when a request is sent to a location to which a class has already been redirected, if it is further redirected from there.

C13-43: The Status field in a response including ClassPortInfo because of redirection **shall** have the MADHeader:Status.RedirectionRequired bit set indicating that a ClassPortInfo attribute was returned rather than the expected attribute.

C13-43.1.1: When a request for a particular MADHeader:MgmtClass has been redirected to another location, that location **shall** continue to service requests for the MADHeader:MgmtClass until either the location becomes inoperable for some reason or the requests are redirected again away from that location.

A response with the MADHeader:Status.RedirectionRequired bit set indicates that the request was not performed and that the request must be issued to the alternate interface specified in ClassPortInfo.

Redirection may be used at any time, so requesters should always be prepared to be redirected.

It is permissible for different requesters for the same management class on a channel adapter, switch, or router to be redirected to a different interface. The redirection operation is depicted in Figure 154 GSI Redirection on page 754.

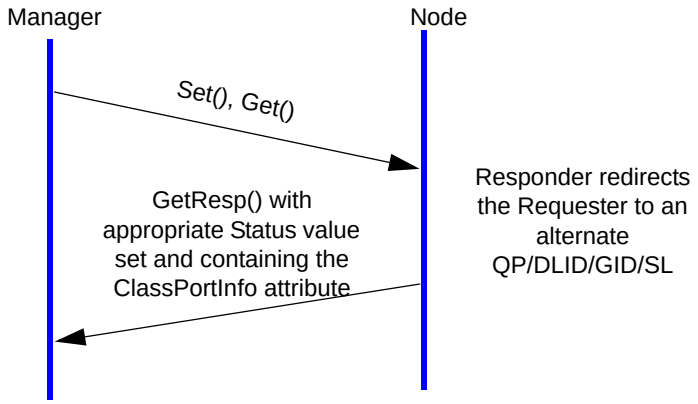


Figure 154 GSI Redirection

Redirection information is also available by doing a normal Get() specifying the class of interest in the MADHeader:MgmtClass field of the MAD header and ClassPortInfo as the attribute.

The ClassPortInfo attribute contains all of the information necessary to access the redirected service either from within the same subnet or from a different subnet. ClassPortInfo may be programmed to include all of the parameters a source needs to form a complete GRH.

C13-44: A GRH **shall** be included in redirected class messages only if the ClassPortInfo:RedirectGID is non zero.

If ClassPortInfo:RedirectGID is non-zero, and redirection is in effect, then a GRH also need not be used if the MAD does not have to exit a subnet.

It is the responsibility of any entity programming ClassPortInfo to assure that the parameters provided for accessing redirected services are consistent with address and access controls applicable to the redirected service.

The ClassPortInfo attribute is described in 13.4.8.1 ClassPortInfo on page 734. The Status field is described in 13.4.7 Status Field on page 731.

13.5.3 MAD VALIDATION

Packets arriving at a port of a channel adapter, switch, or router are validated according to the validation rules specified in 5.2 Data Packet Format on page 151 and 9.6 Packet Transport Header Validation on page 269. Only packets so validated are delivered to management entities. The contents of the data payload are further validated by management entities to validate that the data payload contains a valid MAD.

Valid MADs are delivered to appropriate management entities for processing.

13.5.3.1 MAD VALIDATION FOR SUBNET MANAGEMENT MADS

C13-45: This compliance statement is obsolete and has been replaced by C13-45.1.1:.

C13-45.1.1: Data payloads arriving at the SMI (QP0) **shall** be validated as indicated in the bulleted list below. Packets failing one or more of these check are discarded and no action is taken in response unless the method is a Get() or a Set(). For Get() and Set() methods, the return of a GetResp() when validation has failed is optional.

- The data payload length **shall** be 256 bytes
- LRH:VL **shall** be 15

- BTH:QP **shall** be 0
- BTH:OpCode **shall** be Send only UD
- MADHeader:BaseVersion **shall** be 1
- MADHeader:MgmtClass **shall** specify a class of Subn or Directed Route Subn
- MADHeader:Method **shall** specify a method supported by the class specified in MADHeader:MgmtClass
- MADHeader:AttributeID **shall** specify an attribute supported by the class specified in MADHeader:MgmtClass; and the MADHeader:Method/MADHeader:AttributeID combination **shall** be valid, e.g., the combination of method and attributeID is identified by "X" in the method/attribute map table of a class (e.g., Table 130, "Subnet Management Attribute / Method Map," on page 811).

o13-18: If a GetResp() is returned, any conditions detected that have corresponding codes assigned in Table 115 MAD Common Status Field Bit Values on page 732 **shall** be reflected by corresponding settings of bits in MADHeader:StatusField in the response.

If a channel adapter, switch, or router supports a subnet manager, some MADs may be destined for the SMA while others may be destined for the SM. The discrimination between the SMA as a destination and the SM as a destination is based on the class, the method, and the attribute. See 14.2 Subnet Management Class on page 794. Table 122 SM MAD Sources and Destinations on page 756 indicates which SMPs originate at an SM, which SMPs originate at an SMA, and which SMPs may be destined to SMAs or to SMs.

Table 122 SM MAD Sources and Destinations

MAD Type	Source	Destination	Notes
Get(*)[a]	SM	SMA	Applies for all attributes except SMInfo
Get(SMInfo)	SM	SM	Applies only for the SMInfo attribute
Set(*)[a]	SM	SMA	Applies for all attributes except SMInfo
Set(SMInfo)	SM	SM	Applies only for the SMInfo attribute
GetResp(*)[a]	SMA	SM	Applies for all attributes except SMInfo
GetResp(SMInfo)	SM	SM	Applies only for the SMInfo attribute
Trap()	SMA	SM	Applies to all subnet management traps.
TrapRepress()	SM	SMA	Applies to all subnet management traps.

a. The asterisk is used to indicate any attribute other than SMInfo.

This specification does not require that subnet managers be implemented in any particular way. It does require that subnet managers be able to originate and receive subnet management MADs. However an SM is realized, it must create and receive packets adhering to the on-the-wire formats specified.

C13-45.1.2: The SM **shall** only send packets with a source QP of 0 and and a destination QP of 0, using VL15.

It is recommended that any packet received on QP 0 be discarded if its Source QP is different from zero.

C13-46: The SMI **shall** handle all Directed Route SMPs as described in 14.2.2 SMPs and Directed Route Algorithm on page 797.

13.5.3.2 MAD VALIDATION FOR SUBNET ADMINISTRATION AND GENERAL SERVICES

13.5.3.2.1 MAD VALIDATION AT THE GSI

C13-47: Data payloads arriving at the GSI (QP1) **shall** be validated as specified in the bulleted list below. Packets failing one or more of these checks are discarded and no action is taken in response unless the method is a Get() or a Set(). For Get() and Set() methods, the return of a GetResp() when validation has failed is optional.

- The data payload length must be 256 bytes.
- LRH:VL must not be 15
- BTH:QP must be 1
- BTH:OpCode must be Send only UD
- MADHeader:Baseversion must be 1
- MADHeader:MgmtClass must specify a class supported on the channel adapter, switch, or router.

o13-19: If a GetResp() is returned, any conditions detected that have corresponding codes assigned in Table 115 MAD Common Status Field Bit Values on page 732 **shall** be reflected by corresponding settings of the bits in MADHeader:StatusField of the response.

It is not specified how GMPs passing through the GSI are dispatched to the appropriate class agents that are supported and which are not redirected.

If a class is supported and if redirection has been configured for that class, the response to a request arriving at the GSI containing MADHeader:MgmtClass of a redirected class is to reply with the redirection information for the class as specified in 13.5.2 GSI Redirection on page 753.

On CAs implementing the verbs layer specified in Chapter 11: Software Transport Verbs on page 546, GSMs may be implemented either below the verb layer or above the verb layer. For a GSM implemented above the verb layer and communicating via a source QP1, it is not specified how disambiguation between GMPs destined to GSAs on that CA and GMPs destined to that GSM is performed. The basis for such differentiation is both class and context dependent as well as implementation dependent.

C13-48: If a CA does not support operation of a GSM via QP1 from on top of its verb layer, that is, if it does not implement disambiguation of GMPs destined to a GSA below the verbs and GMPs destined to QP1 associated with a GSM implemented above the verbs, it **shall** not permit QP1 to be created above the verb layer.

See also 13.5.1.2 Processing General Services Management Packets (GMPs) on page 752.

Regardless of the implementation, the behavior of GSAs and GSMs with respect to the injection of messages on the wire, processing of messages from the wire, and responding to messages received must conform to the requirements of applicable sections in Chapter 16: General Services.

Implicitly, implementations of SAs, GSMs and GSAs must be able to send GMPs destined to QP1.

13.5.3.2.2 MAD VALIDATION AT THE SA AND GSAS

Packets arriving at an SA or GSA via QP1 have already been validated as properly formed MADs.

Packets arriving at a SA or GSA via any QP other than QP1 have been redirected. Such packets have not been validated as properly formed MADs.

o13-20: This compliance statement is obsolete and has been replaced by C13-48.1.1:

C13-48.1.1: Agents processing GMPs that have been redirected **shall** first validate the GMPs as follows:

- The data payload length **shall** be 256 bytes
- LRH:VL **shall not** be 15
- The BTH:OpCode **shall** be Send only UD
- MADHeader:BaseVersion **shall** be 1
- MADHeader:MgmtClass **shall** specify a class supported on the channel adapter, switch, or router.

C13-49: This compliance statement is obsolete and has been replaced by C13-49.1.1:.

C13-49.1.1: All packets arriving for processing at an SA or a GSA **shall** be further validated as follows:

- MADHeader:Method **shall** specify a method supported by the class specified in MADHeader:MgmtClass
- MADHeader:AttributeID must specify an attribute supported by the class specified in MADHeader:MgmtClass; and the MADHeader:Method/MADHeader:AttributeID combination must be valid, e.g., the combination of method and attributeID is identified by “X” in the method/attribute map table of the class.

C13-50: GMPs failing one or more validity checks **shall** be discarded unless the method is one for which a response is normally returned (such as Get(), Set(), or Report()). The return of a response when validation has failed is optional.

o13-21: If a GetResp() is returned, any conditions detected that have corresponding codes assigned in Table 115 MAD Common Status Field Bit Values on page 732 **shall** be reflected by corresponding settings of the bits in the status field of the response.

GSAs are not required to check the validity of the attribute content.

Additional class specific checking requirements may be specified. Such requirements, if any, are defined in the class specific sections of Chapter 15: Subnet Administration on page 882 and Chapter 16: General Services on page 930.

13.5.3.3 CONSOLIDATED MAD VALIDATION FLOW DIAGRAMS

The figures which follow provide a consolidated view of the entire validation process for a SMP or GMP MAD. Both pseudo-code and flowcharts are used, depending on which afforded the greatest clarity. For completeness, it includes MAD tests specified in this chapter and Chapter 14: Subnet Management on page 794, as well as the subset of the LRH and GRH checks that apply to MADs. The latter were derived from the checks specified in Chapter 7: Link Layer on page 167, Chapter 8: Network Layer on page 222 and Chapter 9: Transport Layer on page 230; those chapters should be considered authoritative if any discrepancy between this section and those chapters is noted.

The process begins at the start of Figure 155, LRH Check, on page 760.

If
• ICRC and VCRC are good /* C7-11: */
• and LRH:Lver = 0 /* C7-11: */
• and ((LRH:LNH=0b10) & (LRH:PktLen =72)) | ((LRH:LNH=0b11) & (LRH:PktLen =82)) /* C7-11:, C13-3: */
• and LRH:DLID=PortInfo:LID or 0xFFFF /* C7-11: */
• and (VL is operational and PortState = (Active or Armed)) | (VL = 15 and DLID is unicast) /* C7-11: */
• and (VL is not 15) or (LNH indicates IBA local packet) /* C7-11: */
then go to Figure 156, BTH Check, on page 760;
else, drop packet.

Figure 155 LRH Check

If BTH:DestQP= 0 /* C9-37:, C13-33:, C13-45.1.1::*/
then
if
• BTH:Tver=0 /* C9-5:, C9-36: */
• and LRH:LNH!=0b11
then if
• BTH:OpCode=0b01100100 (Send UD only) /* C13-45.1.1: */
then go to Figure 159, SMP Check 1, on page 761
else, go to Figure 157, BTH Check Extension, on page 760
else drop packet.
else if
• BTH:DestQP= 1 /* C9-37:, C13-34:, C13-47: */
then
i
• BTH:Tver=0 /* C9-5:, C9-36: */
• and LRH:LNH!=0b1)1
or [GRH is present (LRH:LNH=0b11) and GRH check succeeds (see Figure 158, GRH Check, on page 761)]
• and BTH:P_Key is valid /* C9-41:, C9-42: */
• and DETH:Q_Key= 0x8001_0000 /* C9-48:, C9-49: */
• and DLID!=0xFFFF /* C9-55: */
then if
• BTH:OpCode=0b01100100 (Send UD only) /* C13-47: */
then go to Figure 169, GMP Check, on page 767
else, go to Figure 157, BTH Check Extension, on page 760
else, drop packet.
else, drop packet.

Figure 156 BTH Check

if Method = Set() or Get() /* C13-45.1.1:, C13-47: */
then
• Optional:GetResp() with Bad Version /* C13-45.1.1:, C13-47: */
• drop packet.
else, drop packet.

Figure 157 BTH Check Extension

If
• GRH:IPVer=6 /* C8-2:, C9-45: */
• and GRH:PayLen=280 /* C8-6: */
• and GRH:NxtHdr=0x1B /* C8-7:, C9-44: */
• and GRH:DGID is my GID /* C8-11:, C9-46: */
then return into Figure 156, BTH Check, on page 760, indicating the check succeeded.
else return into Figure 156, BTH Check, on page 760, indicating the check failed (drop packet)

Figure 158 GRH Check

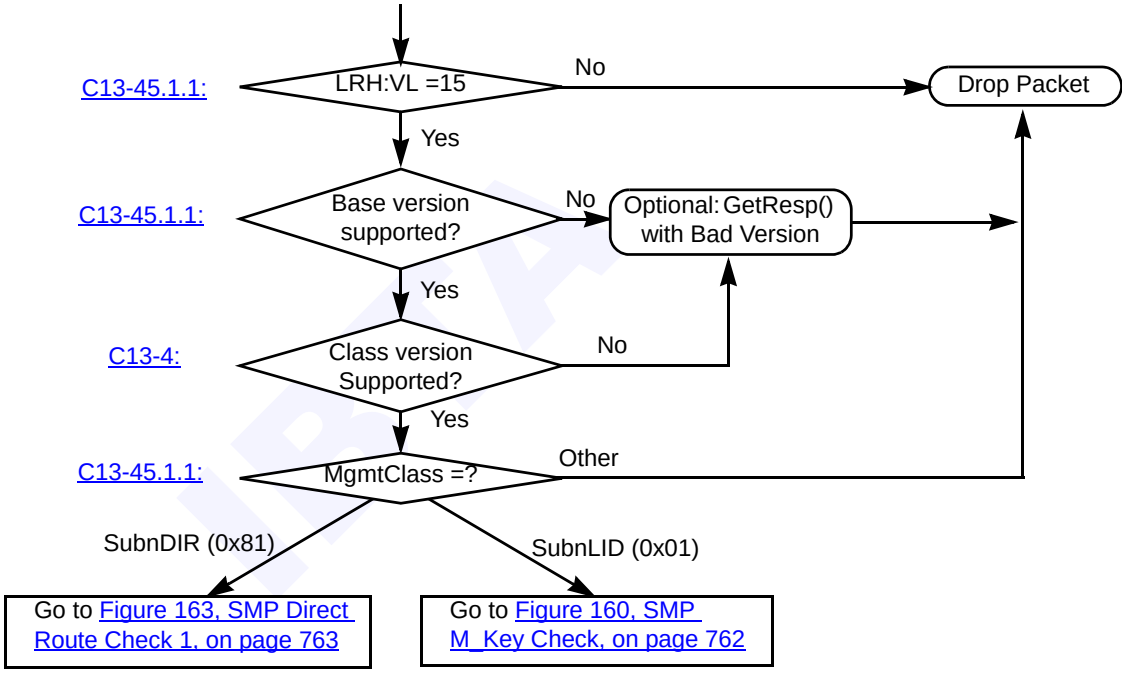


Figure 159 SMP Check 1

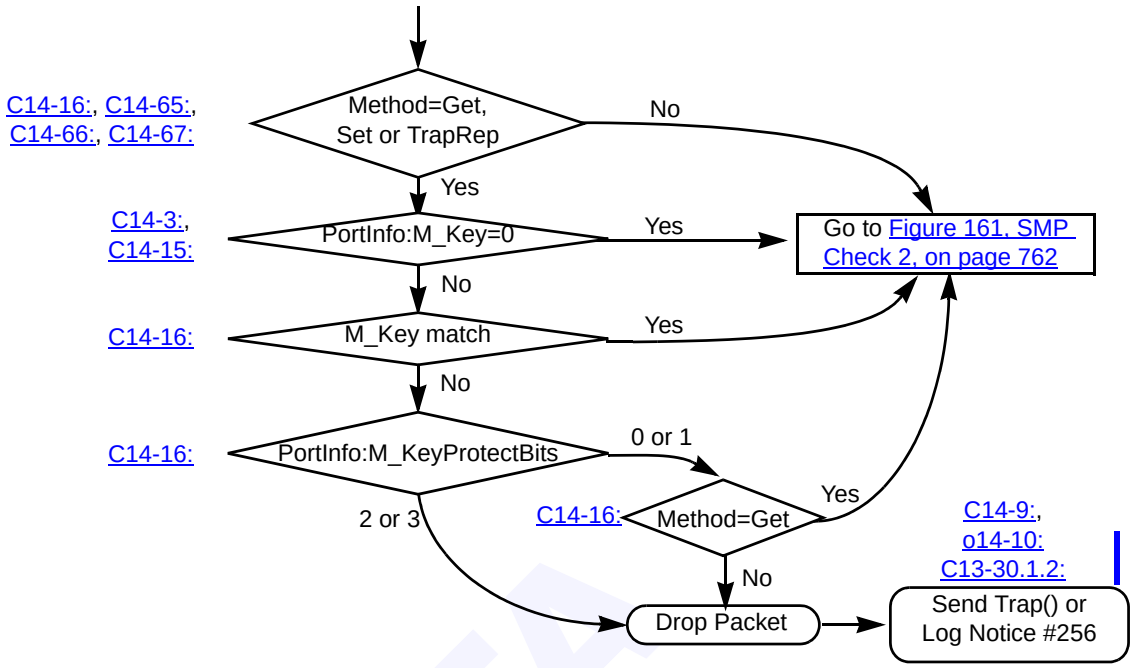


Figure 160 SMP M_Key Check

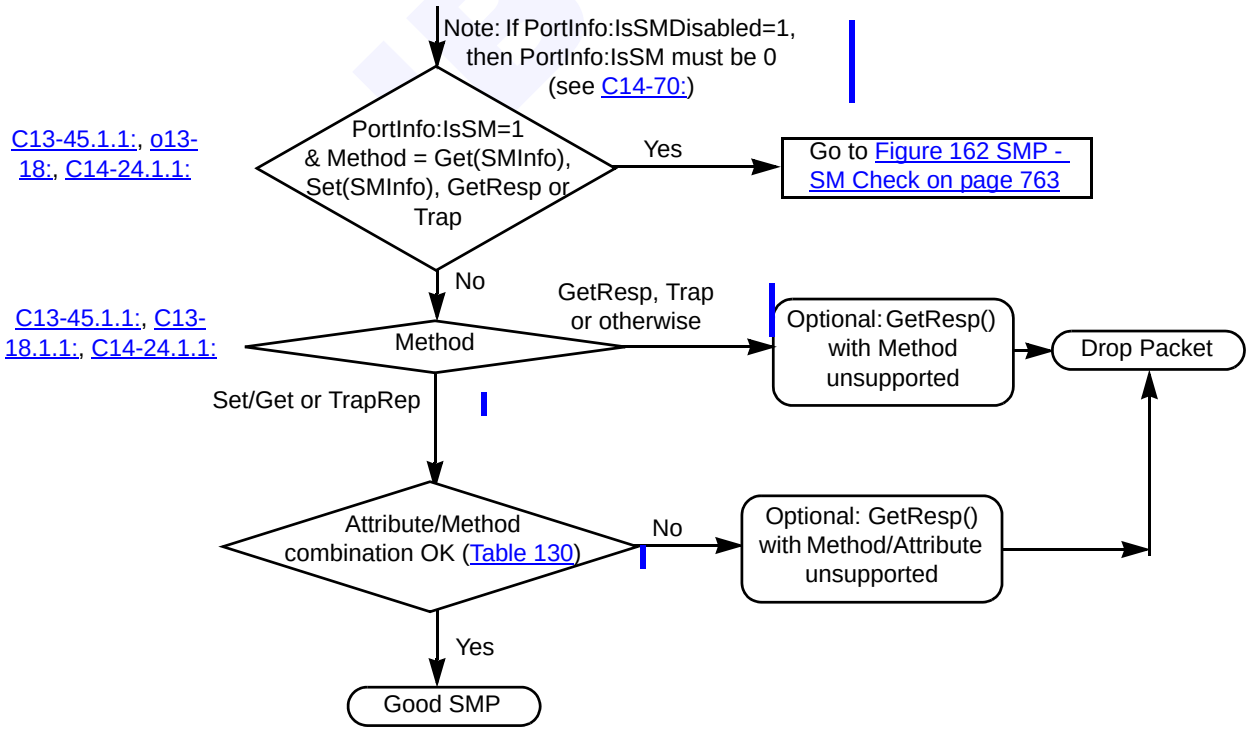


Figure 161 SMP Check 2

If
• Attribute is supported (Table 129 on page 810) /* C14-38:, C14-24.1.1: */
• and AttributeID and Method combination corresponds (Table 129 on page 810)
then GOOD SMP.
else drop packet

Figure 162 SMP - SM Check

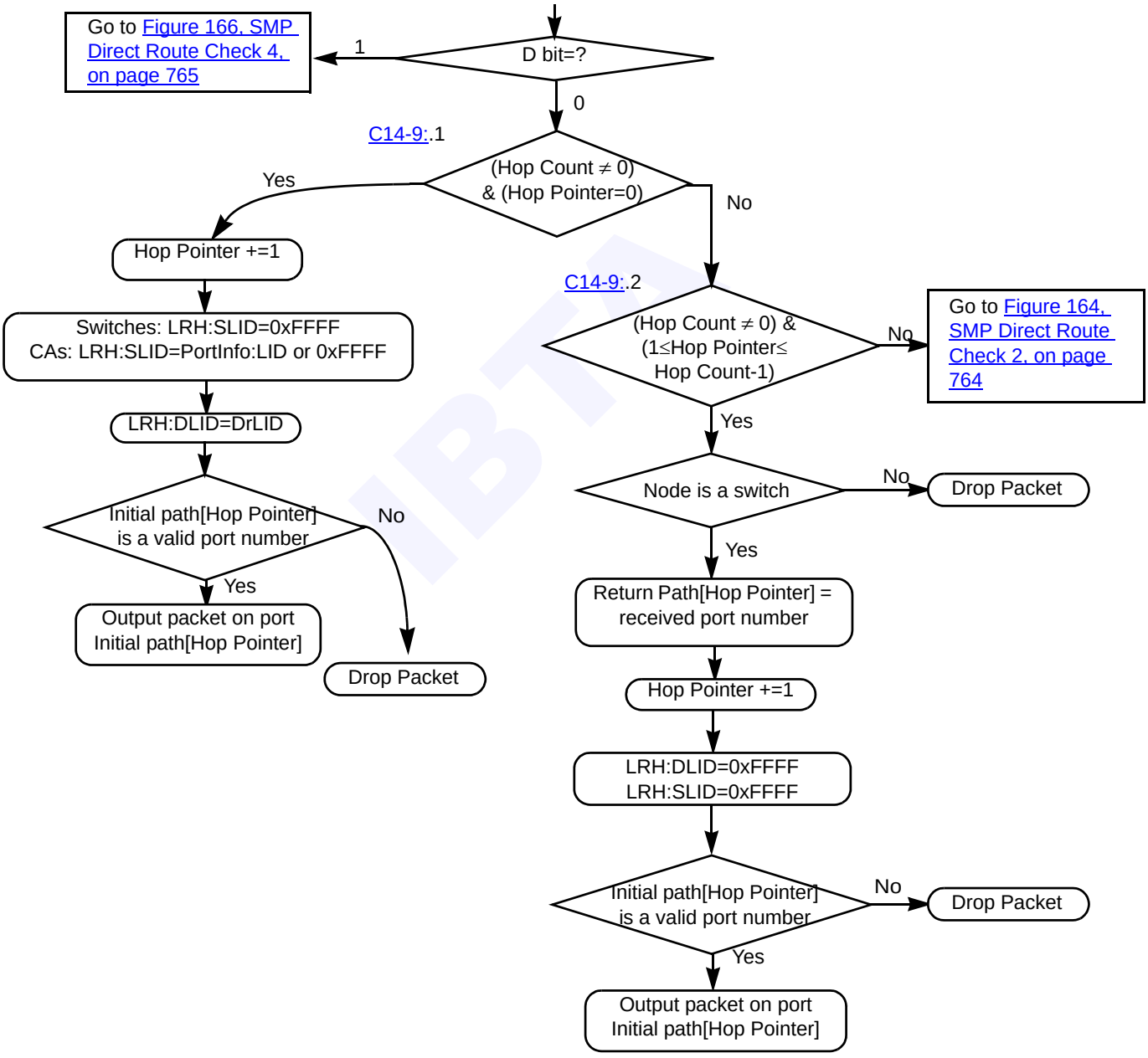


Figure 163 SMP Direct Route Check 1

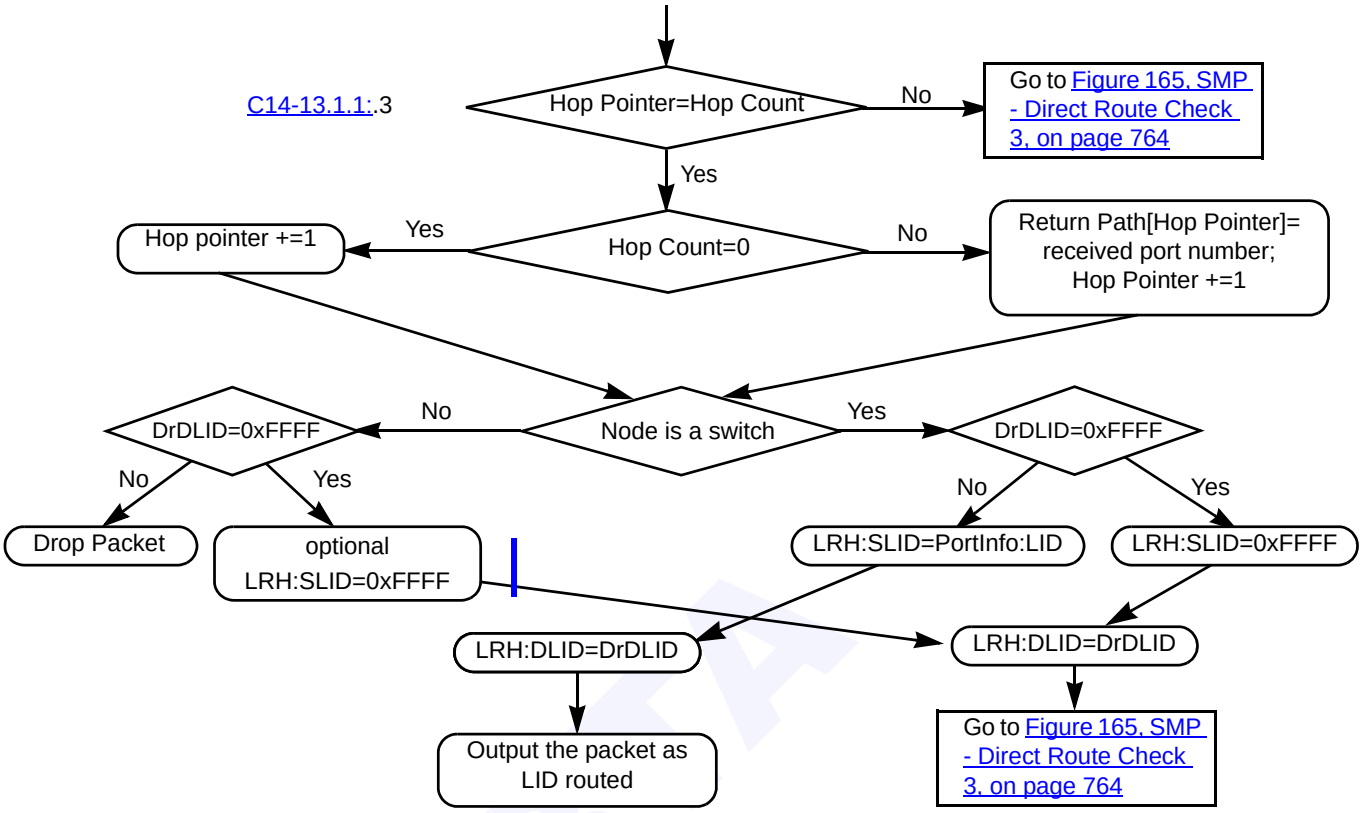


Figure 164 SMP Direct Route Check 2

If
• Hop Pointer = Hop Count +1 (responder node) /*C14-9: 4 */
then go to Figure 160, SMP M_Key Check, on page 762.
else /* this means Hop count + 2 ≤Hop pointer ≤255 */
• drop packet /* C14-9:.5 */

Figure 165 SMP - Direct Route Check 3

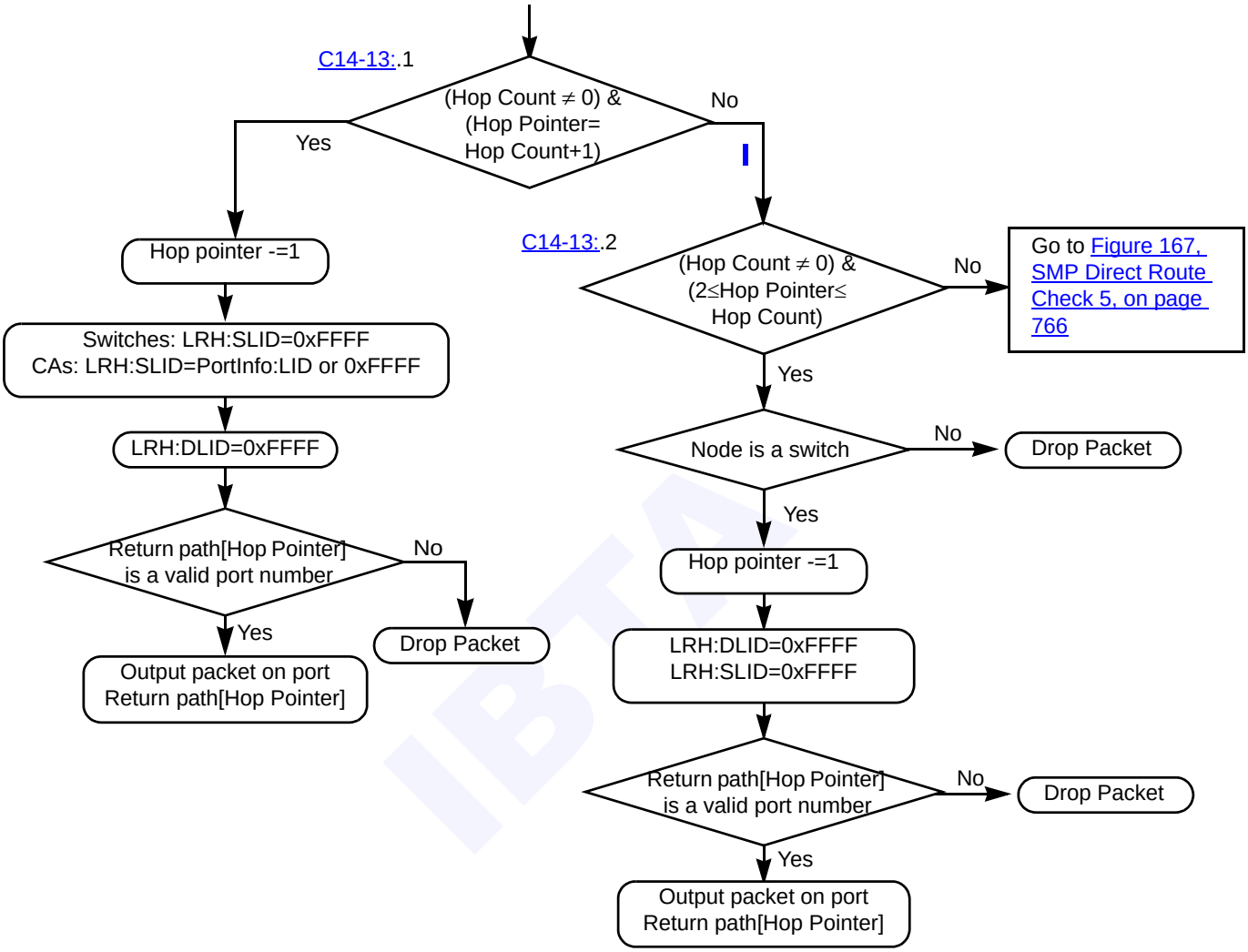


Figure 166 SMP Direct Route Check 4

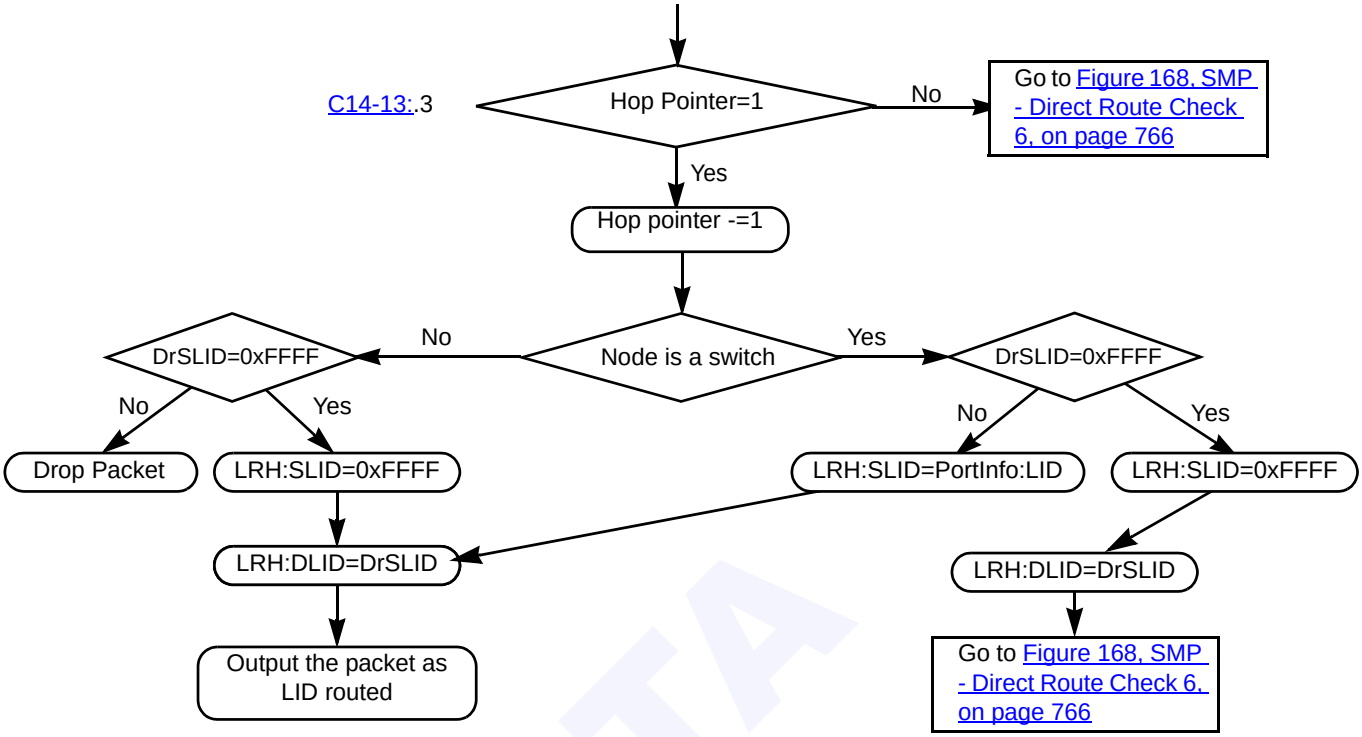


Figure 167 SMP Direct Route Check 5

If
• Hop pointer = 0 (requester node - SM) /* C14-13:.4 */
then go to Figure 160, SMP M_Key Check, on page 762.
else (Hop count + 2 ≤Hop pointer ≤255) drop packet /* C14-3:.5 */

Figure 168 SMP - Direct Route Check 6

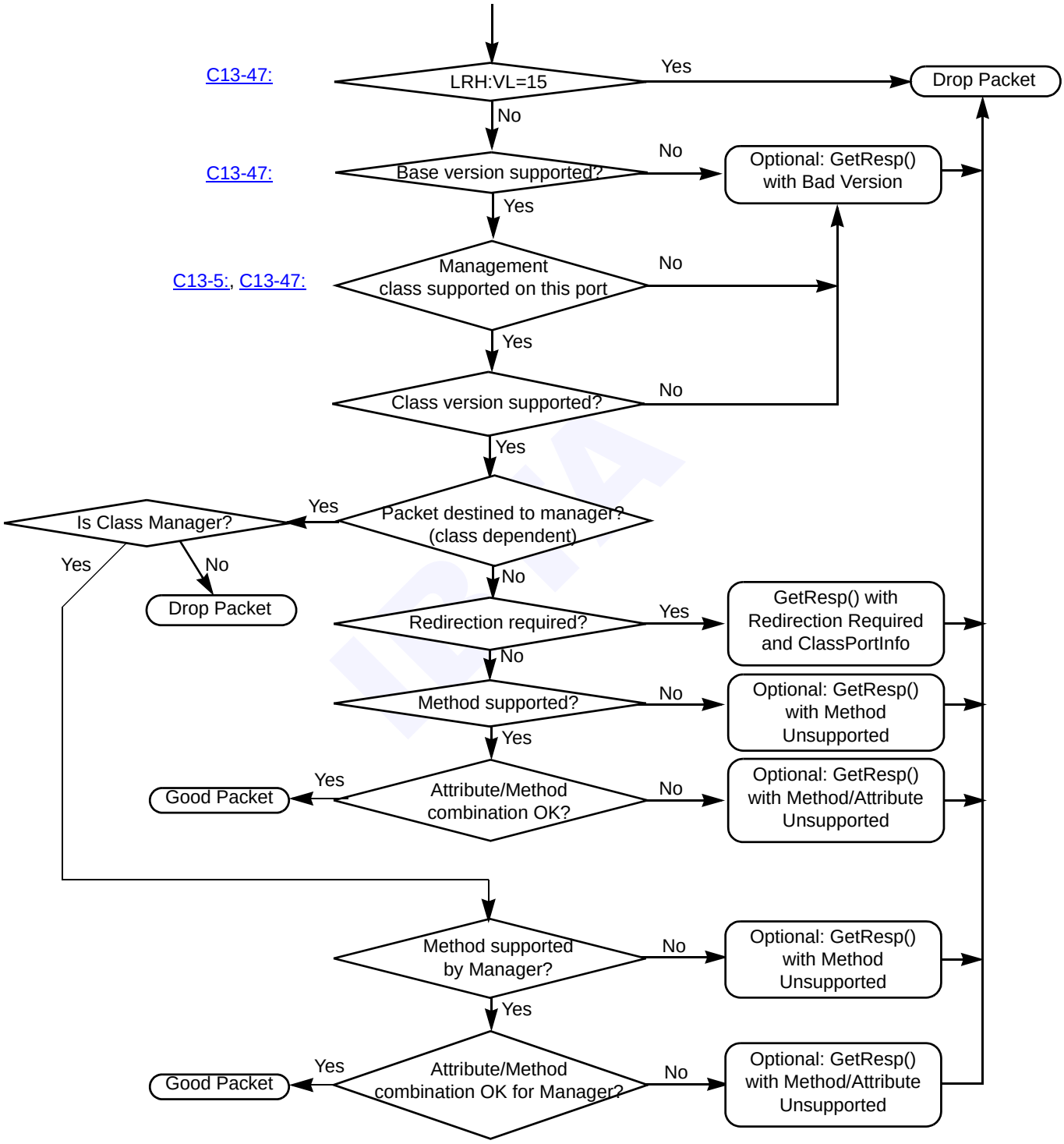


Figure 169 GMP Check

13.5.4 RESPONSE GENERATION AND REVERSIBLE PATHS

Some MAD methods require that the recipient return a response to the sender. Since in many situations the recipient will be unable or not equipped to obtain a path in the reverse direction from Subnet Administration, this requires the existence of a reversible path from sender to recipient. This section discusses reversible paths and response generation.

13.5.4.1 REVERSIBLE PATHS

Loosely, a reversible path is one for which a responder can construct a valid response packet that will reach the original sender using only header information present in the request packet. Not all paths in an IBA fabric are guaranteed to be reversible, and non-reversible paths in some topologies may not be as desirable as reversible paths. However, at least one reversible path must be available between every pair of endports in a subnet (see 14.4.3 Initialization Actions on page 868). The reverse path constructed from the request headers may or may not traverse the same physical links as the forward path and might not be as efficient as a path to the same endport that was explicitly acquired from SA.

The following sections describe how a recipient constructs a response packet received on a reversible path. The sender may be in the local subnet or in a remote one, so a request packet may or may not include a GRH. Note that the initial subsections which follow describe constructing a response in general; MADs are not the only case that may require reversible path response generation. 13.5.4.5 Responses to MADs on page 769 indicates additional specific requirements for responding to MADs and sending requests that require responses.

The precise definition of a reversible path is a path for which the procedure described below will construct a packet that can be successfully transmitted back to the sender.

13.5.4.2 COMMON RESPONSE ACTIONS

A responder always takes the following actions in constructing a response packet:

- The SLID of the received packet is used as the DLID in the response packet.
- The Source QP of the received packet is used as the destination QP in the response packet.
- The SL specified in the received packet is used as the SL in the response packet.
- The responder's P_Key used to match the P_Key in the received packet is used as the P_Key in the response packet.

- Unless it can be determined by some other means that other values apply, the following values are also used:
  - MTU = 256 bytes
  - Rate = 2.5 Gb/s
  - QP = source QP of received packet
  - Q_Key = source Q_Key of received packet

13.5.4.3 CONSTRUCTING A RESPONSE WITHOUT A GRH

If the request packet does not contain a GRH, the response packet does not contain a GRH and is constructed as follows:

- Perform the actions specified in 13.5.4.2 Common Response Actions on page 768.
- Fields not otherwise specified in that section are filled in according to the requirements of the transport service. See Chapter 9: Transport Layer on page 230.

13.5.4.4 CONSTRUCTING A RESPONSE WITH A GRH

If the original request packet contained a GRH, then the response packet must contain a GRH. In this case the response packet is constructed as follows:

- Perform the actions specified in 13.5.4.2 Common Response Actions on page 768.
- Insert a GRH in the response packet with its fields set as follows:
  - The DGID is copied from the SGID in the GRH of the received packet.
  - FlowLabel and TrafficClass are copied from the GRH in the received packet.
  - HopLimit is set to 0xFF.
- Fields not otherwise specified in this section are filled in according to the requirements of the transport service. See Chapter 9: Transport Layer on page 230.

13.5.4.5 RESPONSES TO MADS

C13-50.1.1: A sender of a request MAD requiring a response **shall** send that request MAD using a reversible path.

Reversible paths can be acquired from the SA using PathRecord:Reversible (see 15.2.5.16 PathRecord on page 899).

C13-51: This compliance statement is obsolete and has been replaced by C13-51.1.1:.

C13-51.1.1: If the request MAD does not contain a GRH, the response MAD **shall** be constructed as described in 13.5.4.3 Constructing a Response Without a GRH on page 769 with the addition that, for GMPs received on the GSI or redirected, the well-known Q_Key (0x8001_0000) shall be used in the DETH.

C13-52: This compliance statement is obsolete and has been replaced by C13-52.1.1:

C13-52.1.1: If the request MAD contained a GRH, then the response MAD **shall** be constructed as described in 13.5.4.4 Constructing a Response With a GRH on page 769, with the addition that, for GMPs received on the GSI or redirected, the well-known Q_Key (0x8001_0000) shall be used in the DETH.

13.6 RELIABLE MULTI-PACKET TRANSACTION PROTOCOL

Management Datagrams necessarily use a least-common-denominator form of communication: minimal-MTU (256 byte) Unreliable Datagrams. However, there are circumstances where management classes must reliably transfer data in quantities greater than will fit in a single MAD. To facilitate this, IBA management defines a standard Reliable Multi-Packet Transaction Protocol (RMPP) that may be used by any class needing the ability to transfer large amounts of data (up to 2^{32} packets) in a single logical transaction. This section defines that protocol. The intent of the definition is to allow implementation of a single instance of the RMPP protocol engine that may be used by all management classes present at an endport, freeing the class implementations from considering protocol elements such as retries, acknowledgements, timeout processing, etc.

RMPP is specified in terms of two communicating entities with complementary roles: A *Sender*, which is the source of the data transferred; and a *Receiver* which consumes the data transferred. The entities fulfilling the Sender and Receiver roles may be class managers, management agents, or entities with other relationships to the class (such a clients to a server class). Any of those entities may take on either role, or both roles—performing both at different times or even simultaneously, if multiple simultaneous RMPP transmissions are supported by the protocol implementation (they are allowed by the protocol).

RMPP implements a sliding window recovery protocol that is vaguely reminiscent of TCP: The Sender emits up to a predetermined number of packets (window size). The Receiver moves the window forward by replying to the Sender with acknowledgement messages. The window size always starts at 1; it may be changed by the Receiver to larger values, and later decreased, through a parameter carried by acknowledgement packets. The Receiver implicitly requests retransmissions by ceasing to acknowledge packets; when the Sender times out waiting for an acknowl-

edgement, it begins retransmission starting with the packet following the last one acknowledged.

13.6.1 MANAGEMENT CLASS USE OF RMPP

The specification of a management class that uses RMPP must include the following:

- The RMPP header. This must be present in every packet of the management class, immediately following the MAD Base Format Header, i.e., starting at byte 24 of every MAD of the class (see 13.6.2.1 RMPP Header on page 772). Any additional header information used by the class must follow the RMPP header.
- A specification of the types of transfers used. Receiver-initiated and Sender-initiated transfers are both possible, as is a double-sided RMPP transfer: Sender initiates an RMPP transfer, following which a response is returned as an RMPP transfer in the opposite direction (the two entities switch roles after completing the first transfer). See 13.6.6 Startup Scenarios on page 790.
- A specification of when the RMPP protocol is activated, i.e., when the bit RMPPFlags.Active=1 (see 13.6.2.1 RMPP Header on page 772) and which role (Sender, Receiver) is taken by which entity. RMPP does not specify which methods, attributes, etc., use the protocol. This specification is often most conveniently done by specifying particular methods which activate and use RMPP. (Note, a packet which causes RMPP to begin need not itself be part of the RMPP protocol, i.e., need not have RMPPFlags.Active=1; see, for example, 13.6.4 Ladder Diagram (Example) on page 780 and 13.6.6.1 Receiver-Initiated Transfer on page 790.) While it is not strictly required by RMPP that a particular method always use or initiate RMPP for every transfer, even for those that happen to fit in a single packet, such transfers will work and the specification will be simplified if this is done.
- Ensure that each RMPP transfer sequence be uniquely identified using a MAD Base Header TransactionID that remains constant through the entire transfer.

A common RMPP engine can examine each packet of a class using RMPP as they flow by, ignoring packets with RMPPFlags.Active=0 and pre-processing those with RMPPFlags.Active=1 for the class. That engine recognizes multiple packets belonging to the same RMPP transmission sequence by using the TransactionID (and other elements; see 13.6.5.2 Context & Dispatching on page 783).

The responsibilities of the implementer of a class using the RMPP protocol will vary depending on the facilities made available by a vendor supplying the RMPP protocol engine implementation. However, it is likely that at least part of the RMPP header setup required must be indirectly or directly specified by the implementer.

The implementer of the protocol must adhere to all the compliance statements in this section: 13.6 Reliable Multi-Packet Transaction Protocol on page 770.

13.6.2 RMPP PACKET FORMATS

The packet formats used by RMPP are described here. Note that RMPP does not define or describe any of the fields in the standard MAD header. In particular, the method, attribute, status, and other fields may be used in any way by the class that uses RMPP.

o13-21.1.1: If a management class uses RMPP, it **shall** use the header formats, packet formats, and field definitions described in 13.6.2 RMPP Packet Formats on page 772.

13.6.2.1 RMPP HEADER

o13-21.1.2: If a management class uses RMPP, it **shall** include the RMPP header shown in Figure 170 RMPP Header Layout on page 772 in every MAD of that class type, in the position shown in that table, with the meanings documented in Table 123 RMPP Header Fields on page 773.

o13-21.1.3: If a management class uses RMPP, and RMPPFlags.Active = 0, all other RMPP Header fields **shall** be reserved.

The standard definition of a “reserved” field defined in 13.4.1 Conventions on page 717 is implied in o13-21.1.3:: Set to 0 when sent, and ignored when received.

o13-21.1.4: RMPPVersion **shall** have a value of 1 in all MADs containing an RMPP header with RMPPFlags.Active=1.

The contents of the Standard MAD header, including Method, Attribute, AttributeModifier, and Status, are class-specific.

bytes	bits 31-24	bits 23-16	bits 15-8		bits 7-0
0-23	Standard MAD Header (see Figure 145 MAD Base Format on page 719)				
24	RMPPVersion	RMPPType	RRespTime	RMPPFlags	RMPPStatus
28	Data1				
32	Data2				

Figure 170 RMPP Header Layout

Table 123 RMPP Header Fields

Field	Length (bits)	Description
RMPPVersion	8	Version of RMPP. **Shall** be set to the version of RMPP implemented.
RMPPType	8	Indicates the type of RMPP packet being transferred: • 0: Not an RMPP packet; illegal if RMPPFlags.Active=1. • 1: DATA packet; **shall** be sent only from Sender to Receiver. • 2: ACK packet; **shall** be sent only from Receiver to Sender. • 3: STOP packet; may be sent in either direction • 4: ABORT packet; may be sent in either direction • 5-255: reserved.
RRespTime	5	Encodes a time value in the manner of ClassPortInfo:RespTimeValue (13.4.6.2.1 PortInfo:SubnetTimeout on page 727), with the exception that the value 32 (0x1F) is used to indicate that no time value is provided. The use of RRespTime is discussed in 13.6.3 Timeouts on page 777).
RMPPFlags	3	Flags providing information about the packet: • bit 2: Last. If RMPPType = DATA, 1 indicates that this is the last packet of the transfer; 0 indicates it is not. If RMPPType is not DATA, **shall** be 0. • bit 1: First. If RMPPType = DATA, 1 indicates that this is the first packet of the transfer; 0 indicates it is not. If RMPPType is not DATA, **shall** be 0. • bit 0: Active. **Shall** be 1 on every packet that is part of an RMPP transmission sequence; **shall** be 0 otherwise. See o13-21.1.3: on page 772.
RMPPStatus	8	Status code, used by RMPP to signal error status or lack thereof. The values defined in 13.6.2.2 Status Codes on page 773 **shall** be used.
Data1	32	Meaning varies by RMPPType; see the following subsections
Data2	32	Meaning varies by RMPPType; see the following subsections.

13.6.2.2 STATUS CODES

o13-21.1.5: If a management class uses RMPP, it **shall** use the status codes shown in Table 124 RMPPStatus Codes on page 774 to indicate the conditions described there. If a status code is received with an RMPPType value other than those it applies to, the Receiver **shall** discard the packet, terminate the protocol, and issue an RMPP ABORT packet with status code “Illegal Status” unless the illegal code appeared in an ABORT or STOP packet; in the latter case the Receiver **shall** discard the packet and terminate the protocol without issuing any additional packets.

Note that the status codes listed in Table 124 represent only status of the RMPP protocol itself; other status information may be carried in the Base MAD Header Status field. Also note that in that same table, “transmitter of the packet” refers to the entity that sends the MAD containing the status code, i.e., the transmitter of the packet can be either a Sender or a Receiver.

Table 124 RMPPStatus Codes

RMPPStatus Value	Applies to RMPPTypes	Diagram Abbrev[a]	Name and Description
0	DATA, ACK		**Normal**. No errors or other conditions are being reported. Note that this is not an allowed status value for STOP and ABORT packets.
1	STOP	ResX	**Resources Exhausted**. The transmitter of the packet must terminate the protocol because it has exhausted its resources.
2-117			Reserved; reception of any RMPP packet containing any of these RMPP-StatusValues shall cause the packet's recipient to emit an ABORT packet carrying RMPPStatusValue 124, "Illegal Status."
118	ABORT	T2L	**Total Time Too Long**. The total time taken by the entire transaction exceeded the time allocated.
119	ABORT		**Inconsistent Last and PayloadLength**. A packet was received with RMPPFlag.Last=1, but the total data transferred does not match the Pay-loadLength specified in the first packet of the transfer; or the length of data transferred in a packet equals or exceeds the PayloadLength initially specified, and RMPPLast=0 in that packet.
120	ABORT		**Inconsistent First and SegmentNumber**. A packet was received with RMPPFlags.First = 1 and SegmentNumber not 1, or SegmentNumber=1 and RMPPFlags.First not 1.
121	ABORT	BadT	**Bad RMPPType**. A packet was received containing an illegal RMPPType value.
122	ABORT	W2S	**NewWindowLast Too Small**. An ACK packet was received specifying a new WL value that is less than the current WL value.
123	ABORT	S2B	**SegmentNumber Too Big**. An ACK packet was received for a segment that has not been sent.
124	ABORT		**Illegal Status**. An RMPPStatus code that is undefined or applied to an invalid RMPPType was received by the transmitter of the packet.
125	ABORT	UnV	**Unsupported Version**. The transmitter of the packet does not support the version of RMPP designated by the RMPPVersion value it received.
126	ABORT	TMR	**Too Many Retries**. The vendor-specific limit on the number of retries was exceeded.
127	ABORT		**Unspecified**. An error in the protocol occurred that is not covered by any other status code. It is recommended that vendors use a vendor-specific code rather than "Unspecified" if feasible.
128-191	class-specific		**Class-Specific**. These codes are used to indicate conditions specific to the class using RMPP. Their reception may or may not indicate that the protocol should be terminated. It is recommended that these codes be used only for errors directly related to the class's use of RMPP; errors unrelated to RMPP should be indicated using class-specific encodings of the Status field of the standard MAD header.

Table 124 RMPPStatus Codes (Continued)

RMPPStatus Value	Applies to RMPPTypes	Diagram Abbrev[a]	Name and Description
192-255	STOP, ABORT		**Vendor-Specific**. These codes may be used to indicate vendor-specific errors that force termination of the protocol.

a. This is the abbreviation used in the flow diagrams (13.6.5 Flow Diagrams on page 782) to save space in those diagrams. Codes not used in the diagrams have no abbreviations. Code 0, "Normal," is used where no code is indicated, so its abbreviation is blank.

13.6.2.3 DATA PACKET

The RMPP DATA packet is used to transfer data from the Sender to the Receiver. Its layout is shown in Figure 171 RMPP DATA Packet Layout on page 775.

bytes	bits 31-24	bits 23-16	bits 15-8		bits 7-0
0-23	Standard MAD Header (see Figure 145 MAD Base Format on page 719)				
24	RMPPVersion	RMPPType=DATA	RRespTime	RMPPFlags	RMPPStatus
28	SegmentNumber				
32	PayloadLength				
36-252	TransferredData (class specific)				

Figure 171 RMPP DATA Packet Layout

The **SegmentNumber** field is a unique identifier of the relative position of each packet within a multipacket request or response. RMPP packets with duplicate SegmentNumbers are considered duplicate packets and may be discarded. SegmentNumbers for multi-packet requests and responses begin at segment number 1. If RMPPFlags.First=1, SegmentNumber must be 1, and vice versa. If this is not the case, the Receiver emits an ABORT packet with an RMPPStatus of "Inconsistent First and Segment-Number" and terminates the transfer.

The **PayloadLength** field is valid only for the first and last packet of an RMPP transfer; if RMPPFlags.First=0 and RMPPFlags.Last=0, Payloa-dlength is ignored.

In the first packet of an RMPP transfer (RMPPFlags.First=1), Payloa-dLength may indicate the sum of the lengths, in bytes, of the Transferred-Data fields in all packets of the entire multipacket response; this is done by using a nonzero value for PayloadLength in the first packet. If Payloa-dLength is zero in the first packet of an RMPP transfer, it indicates that the transfer continues until a packet is received with RMPPFlags.Last=1.

Note that PayloadLength counts all the bytes in the TransferredData field of the DATA packet format. If a class has additional header information contained in that area, the bytes used by that header information are counted in PayloadLength, even though they may appear to the class to not be usable data transferred. This applies in all cases, even when no data is actually transferred. For example, if a class has a 20-byte header in the RMPP TransferredData field, and RMPP is used to transfer no data at all, the PayloadLength is nevertheless 20.

In the last packet of an RMPP transfer (RMPPFlags.Last=1), PayloadLength indicates the number of valid bytes in the TransferredData field, allowing data transfers that are not an integral multiple of the length of the TransferredData field. A transfer terminates when either: (a) a packet containing RMPPFlags.Last=1 is received; or (b) a nonzero PayloadLength was given in the first packet of a transfer, and a packet is received containing sufficient TransferredData bytes to equal or exceed the PayloadLength originally provided. If case (b) occurs and RMPPFlags.Last is not 1 for that packet, the Receiver sends an ABORT packet with RMPPStatus of “Inconsistent Last and PayloadLength” and terminates the transfer.

The same packet may have both RMPPFlags.First=1 and RMPPFlags.Last=1. This indicates the case of a single-packet RMPP transfer. In this case PayloadLength must indicate the number of bytes of data contained in the TransferredData field of that packet. PayloadLength may be zero, indicating the case of no data transferred.

13.6.2.4 ACK PACKET

The RMPP ACK packet is used by the Receiver to acknowledge the receipt of data from the Sender, and to signal how many more packets the Receiver is able to accept before the Sender must pause, waiting for another ACK packet. The ACK packet layout is shown in Figure 172 RMPP ACK Packet Layout on page 776.

bytes	bits 31-24	bits 23-16	bits 15-8		bits 7-0
0-23	Standard MAD Header (see Figure 145 MAD Base Format on page 719)				
24	RMPPVersion	RMPPType=ACK	RRespTime	RMPPFlags	RMPPStatus
28	SegmentNumber				
32	NewWindowLast				
36-252	Reserved				

Figure 172 RMPP ACK Packet Layout

The **SegmentNumber** field identifies which packets have been received. A SegmentNumber value of S indicates that the Receiver has successfully received all packets with SegmentNumbers ≤ S. After receiving an

ACK packet with SegmentNumber = S, the Sender will discard any ACK packets with SegmentNumbers ≤ S.

The **NewWindowLast** field indicates the SegmentNumber at which the Sender must pause, awaiting an ACK, before sending more packets. NewWindowLast must be greater than or equal to SegmentNumber and must be greater than or equal to any value of NewWindowLast previously sent in this RMPP sequence.

The SegmentNumber and NewWindowLast values received in an ACK cause the Sender to move the resend window in the transmission sequence; see 13.6.5.1 Context State Variables on page 782.

13.6.2.5 ABORT AND STOP PACKETS

The RMPP ABORT packet is used to signal a protocol violation and terminate the transmission sequence. The RMPP STOP packet terminates the transmission sequence for a reason allowed in the protocol, such as either side of a transfer running out of resources. The formats of these packet types are identical except for their RMPPType value and the RMPPStatus codes used, and some optional data. That format is shown in Figure 173 RMPP ABORT & STOP Packet Layouts on page 777.

bytes	bits 31-24	bits 23-16	bits 15-8		bits 7-0
0-23	Standard MAD Header (see Figure 145 MAD Base Format on page 719)				
24	RMPPVersion	RMPPType=ABORT or STOP	RRespTime	RMPPFlags	RMPPStatus
28	Reserved				
32	Reserved				
36-252	Optional Extended Error Data				

Figure 173 RMPP ABORT & STOP Packet Layouts

The **Optional Extended Error Data** field may be used in conjunction with class-specific and vendor-specific error codes to transmit additional error information. If neither of those two types of codes are present, it is reserved.

13.6.3 TIMEOUTS

RMPP timeouts are based on four timeout values:

- PathRecord:PacketLifeTime: This is the maximum lifetime of a packet transferred from each participant to the other (possibly two different values) along a path. It is contained in the PathRecord attribute

obtained from Subnet Administration that is used to obtain the parameters needed to send packets from one side to the other. (See 15.2.5.16 PathRecord on page 899.)

- PortInfo:SubnetTimeout. This is the maximum lifetime of a packet from an endport to any other endport in a subnet (see 13.4.6.2 Timers and Timeouts on page 727). It may be used instead if the transfer is known to be within a single subnet or cases where the PacketLifeTime is unavailable. However, PacketLifeTime will in general be preferable since it may be smaller and is also valid across subnets.
- ClassPortInfo:RespTimeValue for the entity using RMPP. This is the processing time for the management agent or SA; see 13.4.6.2.2 RespTimeValue on page 728.
- RRespTime. This is a processing time communicated directly in the RMPP header; see Table 123 RMPP Header Fields on page 773.

The above timeouts are used to calculate two timeout periods used by RMPP: the Response Timeout (Resp) and the Total Transaction Timeout (Ttime). Their calculation is described below. Note that Resp can be different for the Receiver and the Sender, since the entities on each side may have different processing times (e.g., different ClassPortInfo:RespTimeValues, when they have ClassPortInfo attributes).

13.6.3.1 RESPONSE TIMEOUT (RESP)

Resp (Response Timeout) is the maximum expected time for a packet to traverse the fabric, be processed on the other side, and a response returned to the original Sender. If it expires without the response being received, it is assumed that either the original packet or the response was lost. Resp is defined as

$$4.096\ \mu\text{sec} \times (2 \times 2^{PLT} + 2^{RTV})$$

where *PLT* is the PathRecord:PacketLifeTime of a port A sending a packet to a port B. The definition of *RTV*, loosely "response time value," is more complex:

- At the start of each RMPP transfer sequence the value of *RTV* depends on the nature of the entity receiving the packet on port B:
  - If that entity has a ClassPortInfo attribute (e.g., is an agent), *RTV* is the entity's ClassPortInfo:RespTimeValue.
  - Otherwise (the entity has no ClassPortInfo or is unreachable within the time constraints), *RTV* is 20, corresponding to a processing time of approximately 4.3 seconds.[21]

21. The value 20 is chosen to be consistent with C13-13.1.1: on page 728, even though the entity being communicated with is not an agent. Communication that is not between managers and agents occurs, for example, between SA and its clients; those clients do not necessarily have ClassPortInfo attributes.

It is recommended that a non-default RRespTime value be provided in Sender-initiated data transfers in order to avoid using the default as described above.

- During the transfer sequence, the entity on port B may change *RTV*, and hence Resp, by encoding a non-default RRespTime value in any RMPP packet; the value received is used to compute Resp for subsequent packets, until and unless it is changed again. Any such adjustments to *RTV* are specific to the RMPP transfer sequence in which they occur; they do not affect any other sequence.

Resp is used by the Sender when sending a DATA packet and expecting an ACK in return at the end of a transmission window. See Figure 179, RMPP Receiver Termination Flow, on page 787 for the Sender behavior when this time period expires.

13.6.3.2 TOTAL TRANSACTION TIMEOUT (TTIME)

Ttime (Total Transaction Timeout) is a timeout of the entire duration of an RMPP data transfer sequence. It is used by the Receiver to abort an RMPP sequence in the event that the Sender becomes unresponsive. Ttime is calculated by the Receiver after it receives the first DATA packet of the sequence, whose PayloadLength field may contain the total number of data bytes to be transferred (see 13.6.2.3 DATA Packet on page 775). If the PayloadLength is provided in that first packet, Ttime is calculated as[22]

$$4.096\ \mu\text{sec} \times 8 \times \left\lceil \frac{Payloadlength}{220} \right\rceil \times (2^{Ps} + 2^{Rr} + 2^{Pr} + 2^{RTV})$$

where:

- the notation $\lceil \ldots \rceil$ indicates rounding up the division
- *Ps* is the PacketLifeTime of the path used from Sender to Receiver
- *Pr* is the PacketLifeTime of the path used from Receiver to Sender
- *Rr* is the Receiver's own RespTimeValue
- *RTV* is the sender's response time value defined in 13.6.3.1 Response Timeout (Resp) on page 778.
- 220 is the number of data bytes transferred in a single RMPP DATA packet
- 8 is a multiplier to allow for lost packet retransmission.

If no indication of the PayloadLength is present, the class may define its own timeout. If that is not done, a default of 40 seconds is used.

22. This formula is intended to represent the total time to transfer a sequence of PayloadLength bytes with a window size of 1 and a multiplier of 8 to allow for retries. The use of larger window sizes will decrease the transfer time significantly.

13.6.4 LADDER DIAGRAM (EXAMPLE)

Figure 174 RMPP Example on page 780 illustrates the packet flow for a Receiver-initiated RMPP transfer.

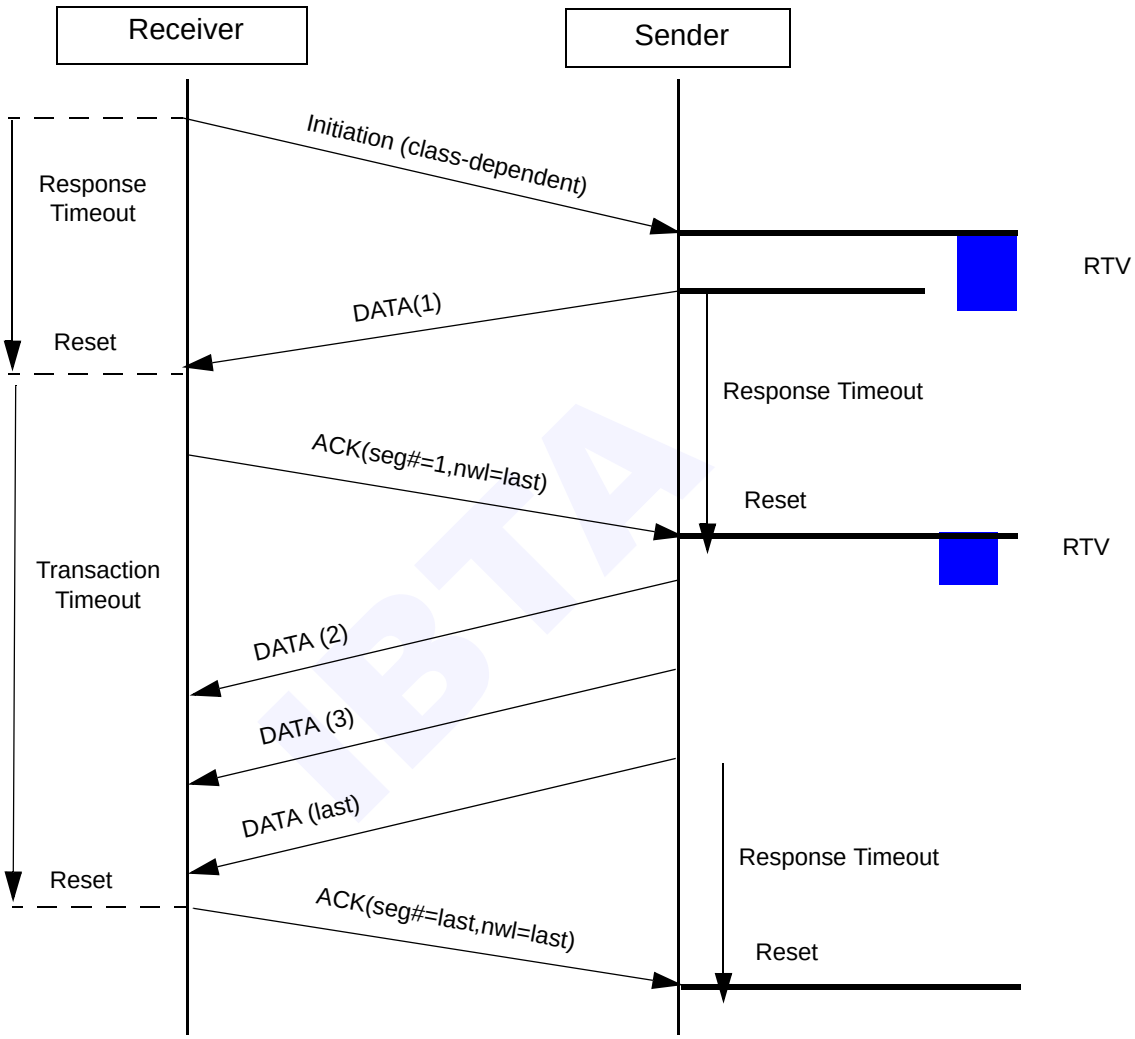


Figure 174 RMPP Example

In Figure 174, the Receiver initiates the multi-packet protocol by first (not shown in the diagram) creating a receive context for the transfer (see 13.6.5.1 Context State Variables on page 782) and then sending a packet whose method or other characteristics informs the Sender class what data to transfer and the TransctionID to use for the transfer. That packet is not itself part of the RMPP transfer; its RMPPFlags.Active bit is 0.

When the request packet is sent, the Receiver initializes the response timer for the transaction, using the initial Response Timeout (described in 13.6.3.1 Response Timeout (Resp) on page 778). If the Receiver does not receive a response packet after the response timer expires, it may retry

the request a vendor-specific number of times; after all retries are exhausted, it terminates.

When the Sender has the data to send, it sends the first DATA packet; this has SegmentNumber=1 and RMPPFlags.First=1, and may contain a PayloadLength indicating the size of the total transfer. Since each transmission sequence begins with a window size of one, the Sender then waits for an ACK packet, using its Resp timeout for this transfer. If an ACK packet is not received in time, it may retry, ABORTing if retries fail.

When Receiver receives the initial DATA packet, it resets its original timer set for receiving the initial DATA packet and it sets a timer using the transaction timer (described in 13.6.3 Timeouts on page 777) for the entire RMPP transaction. It then responds with an ACK packet that acknowledges receipt of SegmentNumber 1 and informs the Sender of a new window size, picking a value the Receiver deems appropriate; for example, it may allow sending a number of packets equal to the number of receive work requests it has enqueued for the transfer. In Figure 174, for simplicity, the window size is set large enough to encompass the entire rest of the transfer. If it were not, additional ACKs and pauses between sending of DATA packets by the Sender would be necessary.

On receiving the ACK, the Sender begins transmission of a stream of DATA packets, starting with SegmentNumber=2. Each time a DATA packet is received, the Receiver can optionally send an ACK packet acknowledging the receipt of DATA packets up to the SegmentNumber received (not illustrated.)

The Sender continues sending packets until all the packets up to the window size specified by the Receiver have been sent. In Figure 174, this would be all the packets to be sent. The last packet sent has RMPPFlags.Last=1, and PayloadLength set equal to the amount of data contained in the last packet. After sending the last packet in the window, the Sender initializes its timeout to Response Timeout and waits for an ACK packet.

The Receiver sends an ACK at the end of the window, indicating that the final packet of the window was successfully received. If this were the last packet sent, the Receiver resets its transaction timer, marks the transaction as successfully completed and sets a timer to delete the context after Response Timeout (described in 13.6.3.1 Response Timeout (Resp) on page 778) to ensure that any subsequent resends by the Sender (due to lost ACK packets) does not result in the Sender incorrectly assuming that the transaction completed in error.

If an ACK is not received within the Sender's timeout, the Sender resends the DATA packets from the last acknowledged packet up to the end of the current window, with ABORT if the retries fail.

If the Receiver does not receive all the packets in this transaction within its transaction timer, it ABORTs the transaction and terminates

13.6.5 FLOW DIAGRAMS

The flow diagrams which follow define the characteristics of RMPP that are constant across all classes using this facility.

In order to be sufficiently detailed, this description contains constructs such as explicit state variables, assumptions about threading and state, etc. It is neither required nor expected that any implementation implement these literally. Rather, these constructs are intended only as a way to specifically describe the external behavior of the protocol to ensure on-the-wire interoperability among implementations.

13.6.5.1 CONTEXT STATE VARIABLES

The description assumes that each RMPP transaction operates in a context, meaning a collection of state variables specific to that transaction whose settings are modified during the course of the operation. Each set of such variables is associated with a tuple of {TID, GID, class} or {TID, LID, class} (see 13.4.6.4 TransactionID usage on page 731), which is constant over the course of each RMPP transmission sequence and serves to uniquely identify that sequence. The number of such contexts simultaneously maintained is vendor-dependent; a minimum of one is required to perform the protocol. Both the Sender and the Receiver will in any implementation contain state variables representing information beyond what is specified here.

The descriptions of window and state variables below refer to the SegmentNumber field of RMPP DATA and ACK packets, often abbreviated "*seg#*" in the diagrams and text of this section.

The state variables relating to the retry window are WF, WL, and NS on the Sender side; and ES on the Receiver side. The relationship between these and the current window is illustrated in Figure 175 RMPP Window State Variable Relationships on page 783.

- **WF** = Window First: Number of the segment that is the first packet in the current window.
- **WL** = Window Last: Number of the segment that is the last packet in the current window.
- **NS** = Next Segment number. The number of the segment that will be transmitted next.
- **ES** = Expected Segment number. The seg# that the Receiver expects to receive in the next packet containing data.

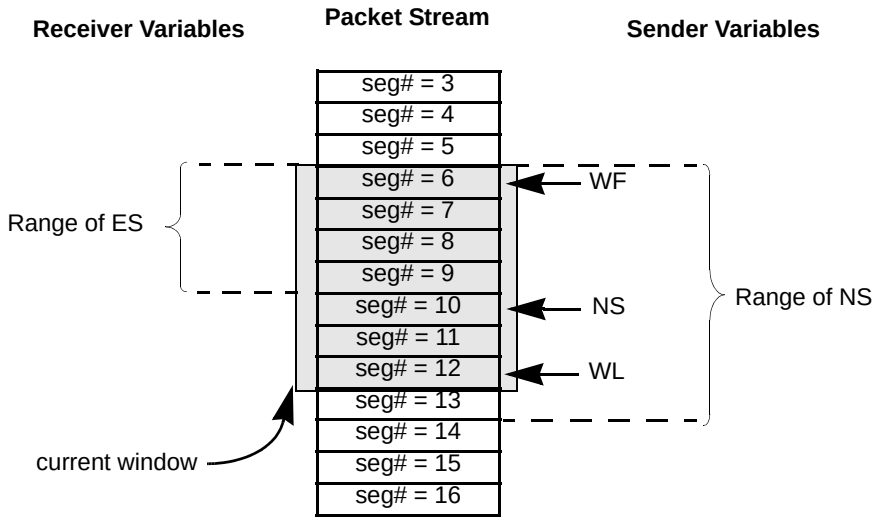


Figure 175 RMPP Window State Variable Relationships

If the protocol is operating correctly, it is always the case that WF≤WL, WF≤NS≤(WL+1), and WF≤ES<NS. Should either side determine that those relationships are violated, the protocol terminates either by explicitly sending an ABORT packet or by timeout. For example, if the Sender receives an ACK acknowledging a packet that has not yet been sent, implying ES≥NS, an ABORT is sent to the Receiver by the Sender.

Additional state variables used are:

- **IsDS** = Is Double-Sided. This is a boolean value (True or False) indicating whether the context is being used as the first half of a double-sided transfer (discussed in 13.6.6.3 Sender-Initiated Double-Sided Transfer on page 792). It is present in both Receiver and Sender contexts.
- **Resp** = The current Resp timeout value, defined in 13.6.3.1 Response Timeout (Resp) on page 778. It is present in both Receiver and Sender contexts.
- **Ttime** = The current total transaction timeout value, as defined in 13.6.3.2 Total Transaction Timeout (Ttime) on page 779. It is present only in the Receiver context.

13.6.5.2 CONTEXT & DISPATCHING

Figure 176 RMPP Dispatcher on page 784 illustrates the operation of the top-level dispatcher that resumes, and in some cases creates and starts RMPP operation within particular contexts.

o13-21.1.6: A class implementing RMPP **shall** exhibit the external, on-the-wire behavior implied by Figure 176 RMPP Dispatcher on page 784.

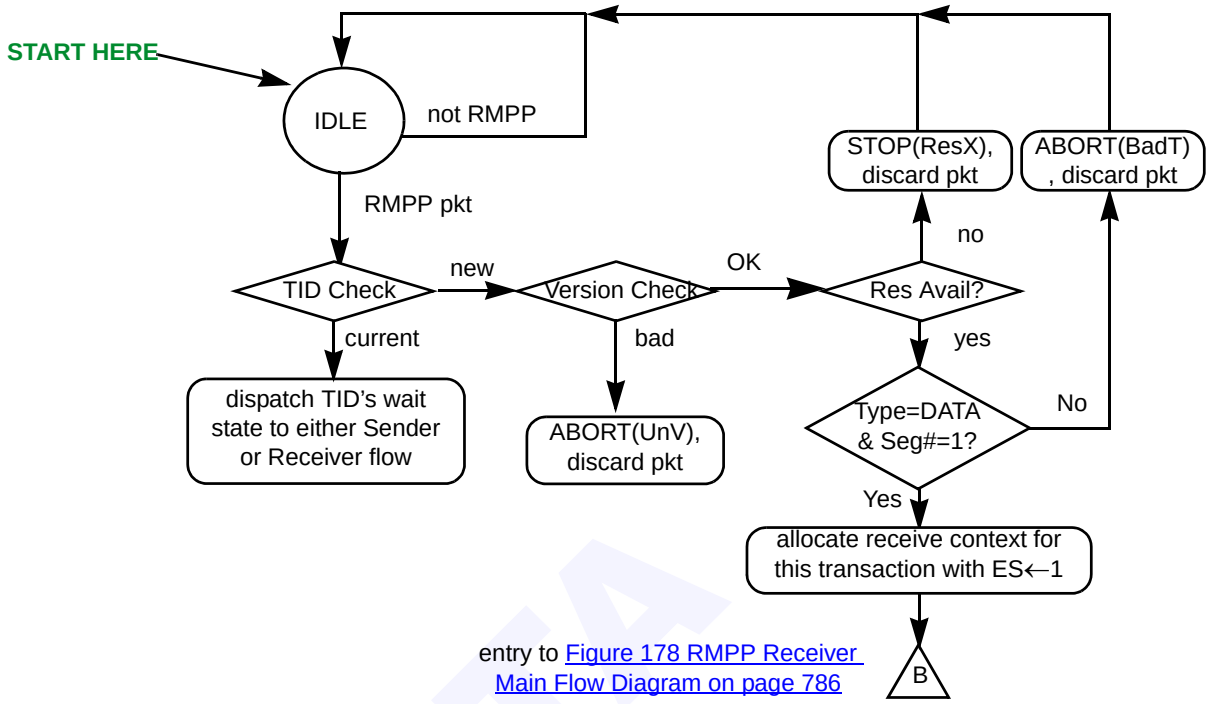


Figure 176 RMPP Dispatcher

Entry to Figure 176 is at the point labelled START HERE.

The following abbreviations are used in Figure 176:

- **Res Avail?** = the resources are available to allocate a new Receiver context.
- **TID Check** = combination of TID, SLID or SGID, and class of a packet identifies it as being part of a transfer that is either in process (current), or has been completed or terminated (old) or is neither of those two cases (new).
- **Version Check** = determines if the RMPPVersion in the packet indicates a version supported by this implementation.

When either the Receiver or the Sender protocols enter a wait state, they implicitly return to the IDLE state in the dispatcher.

If RMPP packets for a transmission sequence are received while that sequence's processing is under way—after it has been dispatched and before it has re-entered a wait—those packets are implicitly queued in the order in which they are received. When the processing of that sequence re-enters the wait state, it is re-dispatched with the next waiting packet.

If RMPP packets are received for a transmission sequence that is not currently executing, but another transfer is currently executing, a separate thread and/or context may be created and dispatched; whether this is done is implementation-specific and may depend on the resources available.

When any currently in-process transmission sequence returns to its wait state, any other may be dispatched with a waiting packet. While the order of packets within any given transfer is preserved, ordering across sequences need not be preserved.

13.6.5.3 COMMON TERMINATION FLOW

Both the Sender and the Receiver use a common flow as their final termination operation to wait for any packets that may have been delayed in the network and as a result arrive after the protocol has terminated. That flow is shown in Figure 177 RMPP Common Termination Flow on page 785.This flow is entered from several others, and is implicitly required whenever they are required. However, note that while Figure 177 is not used for non-error termination in the Receiver case; it has its own termination for that situation.

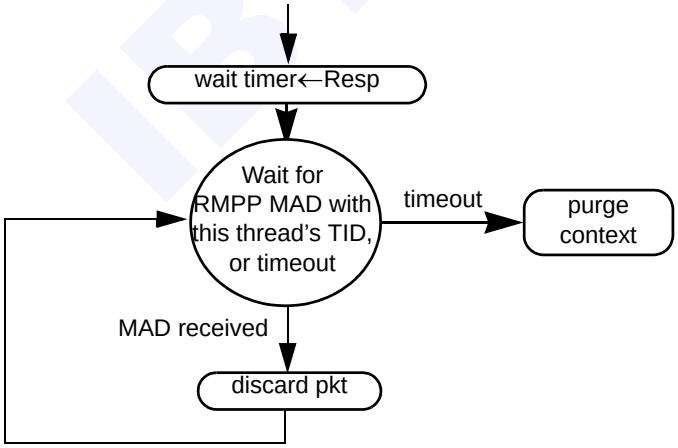


Figure 177 RMPP Common Termination Flow

Note that by the time the flow of Figure 177 is entered, the transaction has completed. Receipt of an ABORT therefore has no effect; ABORT packets are discarded like all other packets.

13.6.5.4 RECEIVER FLOW DIAGRAM

Figure 178 RMPP Receiver Main Flow Diagram on page 786 shows the Receiver logical flow. In addition to what is shown there, Receiver can send a STOP message at any time and terminate this transaction. The processing at the end of a successful Receive operation is shown in

Figure 179 RMPP Receiver Termination Flow on page 787; it is separate from Figure 178 only for convenience of representation.

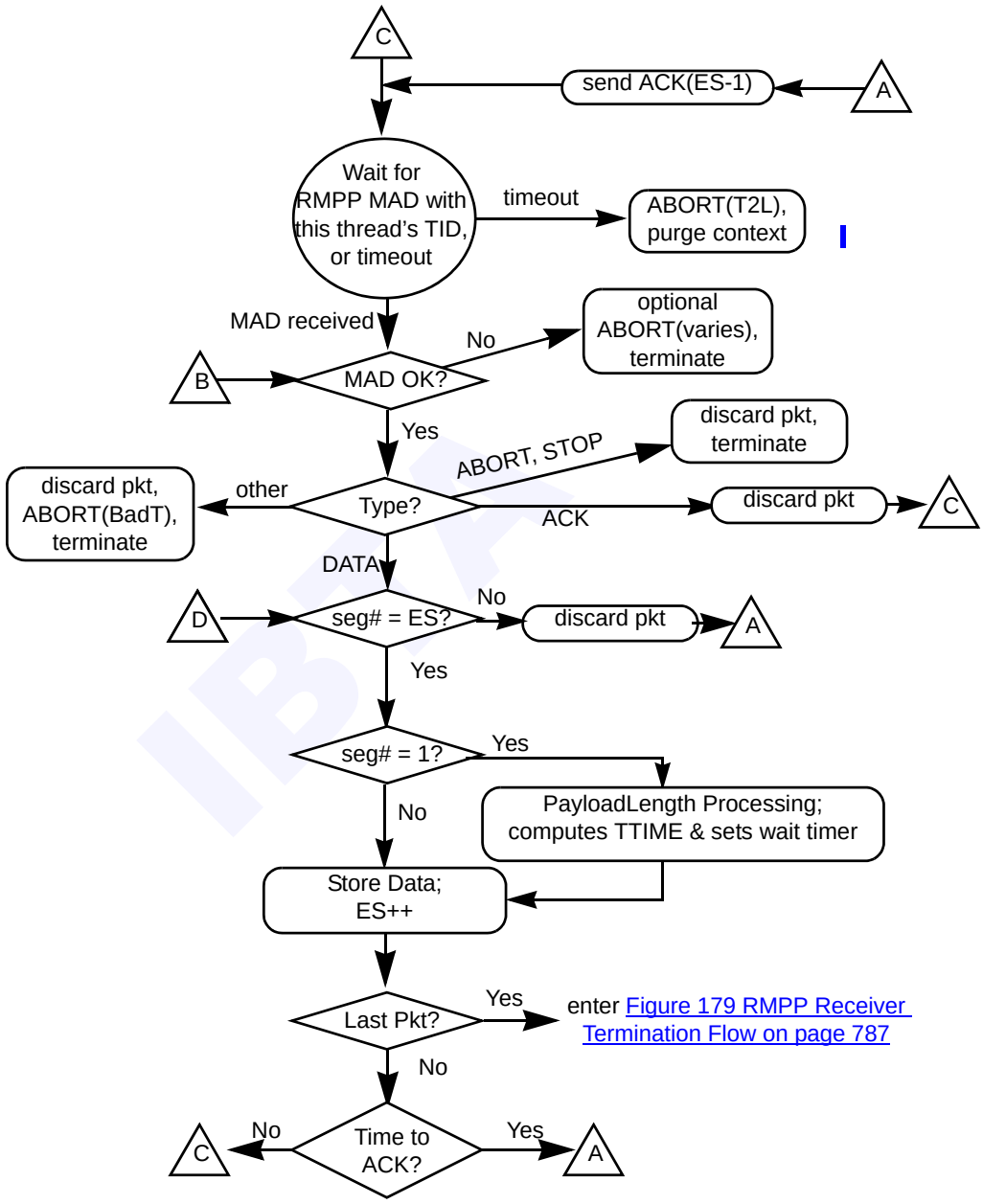


Figure 178 RMPP Receiver Main Flow Diagram

o13-21.1.7: A class implementing RMPP **shall** exhibit the external, on-the-wire behavior implied by Figure 178 RMPP Receiver Main Flow Diagram on page 786.

Entry to Figure 178 is at the points labelled B, C or D, depending on how the transfer is initiated.

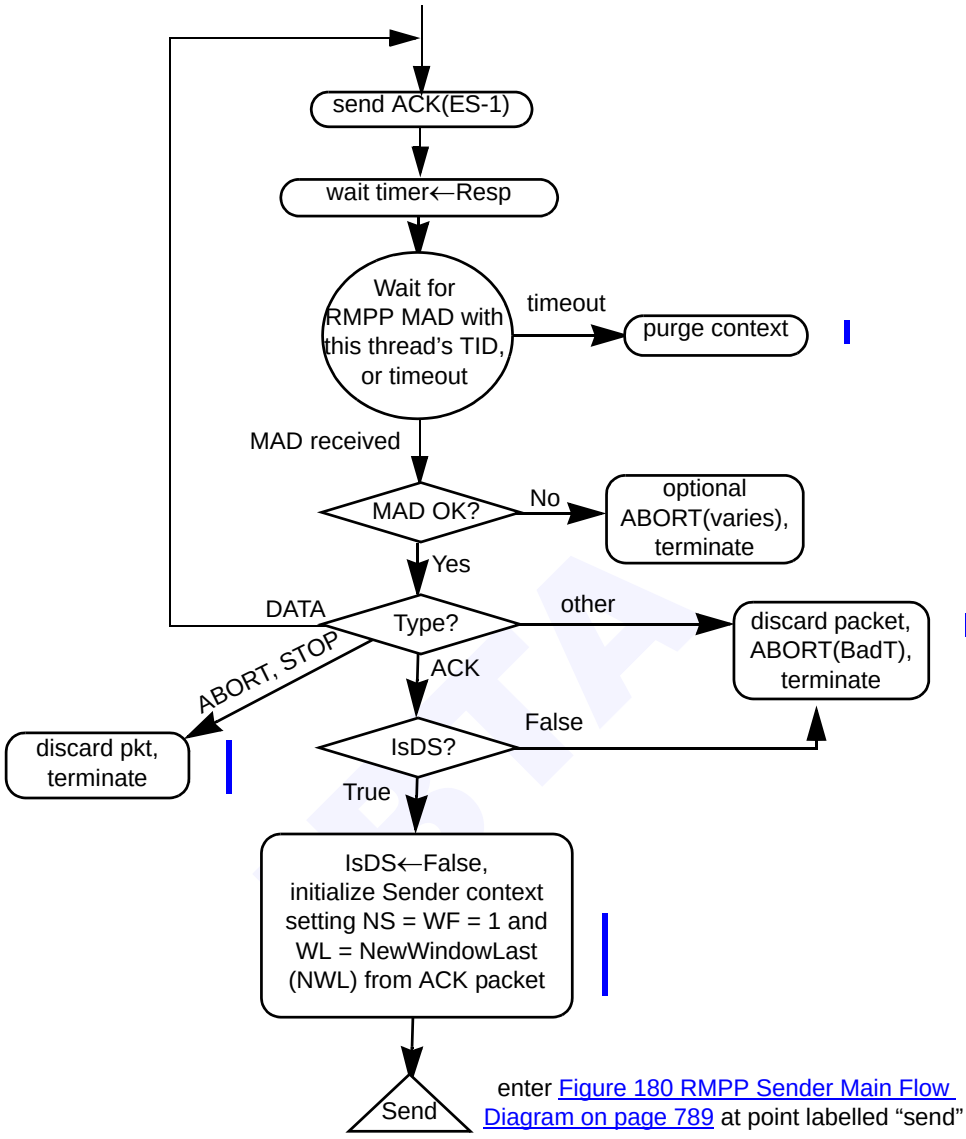


Figure 179 RMPP Receiver Termination Flow

Abbreviations and notation used in Figure 178 and Figure 179:

- **terminate** = enter Figure 177 RMPP Common Termination Flow on page 785. This is used in Figure 178 only to terminate in error cases situations; normal termination enters Figure 179.
- **MAD OK?** = partly class-specific test that various characteristics of the received MAD are appropriate. E.g.: Status in the Standard MAD Header is a value expected; method is a method expected; attribute is an attribute expected; etc. In addition, if the RRespTime value in

the packet is not the default value, the Resp timeout for this sequence is adjusted accordingly (see 13.6.3.1 Response Timeout (Resp) on page 778).

- **Type?** = dispatch based on RMPPType field.
- **Last Pkt?** = based on either the RMPPFlags.Last value or on the payload length, the packet just received is determined to be the last packet of the transaction. Consistency of PayloadLength and flag settings is checked here and may result in an ABORT packet being sent; see discussion in 13.6.2.3 DATA Packet on page 775.
- **Time to ACK?** = the Receiver decides to send an ACK of the last-received segment. This must be done if the packet was the last in the current window; it may also be done at any time prior to that (vendor-specific).
- **Store Data** = the data transferred in the last-received packet is stored and/or transferred to the receiving agent or manager. If this was the last packet of a transmission, the PayloadLength indicates the amount of data contained in the packet; otherwise the entire data part of the packet (class-specific) is transferred.
- **Abort(*value*)** = an RMPP packet is sent with RMPPType=ABORT and RMPPStatus=***value***.
- **ACK(ES-1)** = an RMPP packet is sent with RMPPType=ACK, SegmentNumber=ES-1, and NewWindowLast = what the Receiver desires to indicate as the length of the current window.

13.6.5.5 SENDER MAIN FLOW DIAGRAM

Figure 180 RMPP Sender Main Flow Diagram on page 789 shows the Sender logical flow. In addition to what is shown there, Sender can send a STOP message at any time and terminate this transaction.

o13-21.1.8: A class implementing RMPP **shall** exhibit the external, on-the-wire behavior implied by Figure 180 RMPP Sender Main Flow Diagram on page 789.

Abbreviations and notation used in Figure 180:

- **terminate** = enter Figure 177 RMPP Common Termination Flow on page 785.
- **MAD OK?** = identical to the “MAD OK?” processing of 13.6.5.4 Receiver Flow Diagram on page 785. The class-specific tests may, however, be different.
- **Retry?** = vendor-specific number of retries has not been exceeded.
- **send DATA, inc NS** = send one or more DATA packets and increment NS accordingly. E.g., enqueue send requests for DATA packets that transfer data using successive SequenceNumbers, incrementing

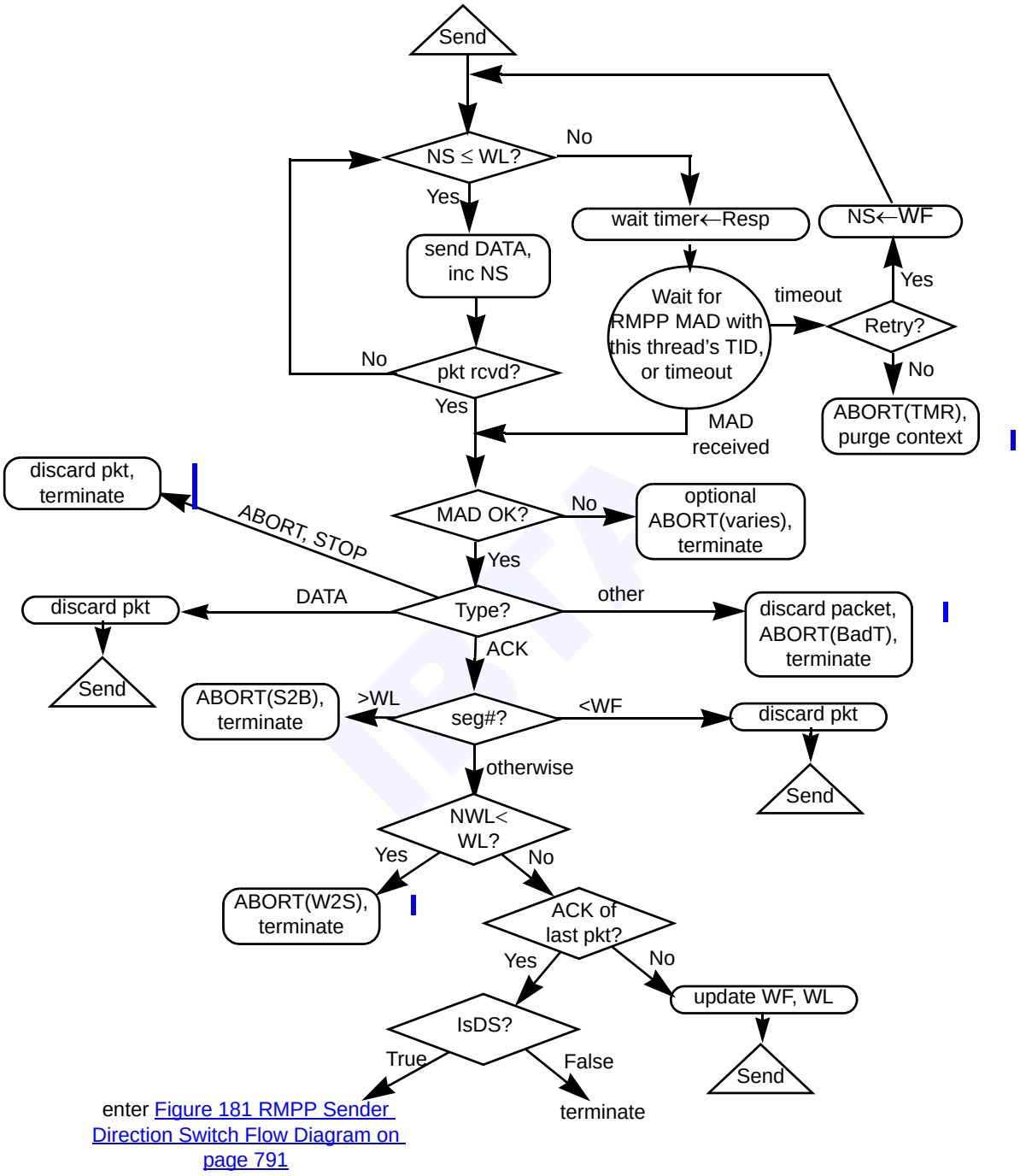


Figure 180 RMPP Sender Main Flow Diagram

NS by 1 for each enqueued request. The sending (or enqueueing) process should stop when either (a) NS reaches WL+1; or (a) all the data for this transmission sequence has been sent.

- **update WF, WL** = set WL to NewWindowLast in the incoming ACK packet and WF to SegmentNumber+1.
- **Type?** = dispatch based on RMPPType field.
- **seg#?** = dispatch based on the value of the segment # field in the MAD.

13.6.5.6 DIRECTION SWITCH

A switch from Sender to Receiver, and vice versa, is required as part of a double-sided transfer (see 13.6.6.3 Sender-Initiated Double-Sided Transfer on page 792). The basic sequence for this direction switch is as follows (ignoring errors, retries, etc.):

- The original Receiver, on receipt of the final DATA packet of the initial sequence, sends an ACK of that packet (true whether it's double-sided or not).
- The original Sender, on receiving that ACK, responds with an ACK indicating a SegmentNumber of 0 and whatever window size it wishes to use initially; it then enters the Receiver Main Flow, becoming a Receiver, and begins receiving packets normally.
- The original Receiver, on receiving an ACK from the original Sender, enters the Sender Main Flow, becoming a Sender, and begins sending packets normally.

Figure 181 RMPP Sender Direction Switch Flow Diagram on page 791 provides the details of the Sender direction switch operation. The Receiver direction switch flow is contained in Figure 179 RMPP Receiver Termination Flow on page 787.The abbreviations and notation used in Figure 181 are the same as those used in prior figures.

13.6.6 STARTUP SCENARIOS

This section illustrates how RMPP transfers may be initiated. Other techniques than those described may be used; this is in general class-dependent.

13.6.6.1 RECEIVER-INITIATED TRANSFER

A Receiver may effectively initiate an RMPP transfer by sending a single-packet MAD which the receiving class interprets as a request for the initiation of an RMPP transfer. For example, this is what the GetTable() method of SA does.

A way for the Receiver to do this is to perform the following sequence. (Issues of allocating receive buffers, posting receive work requests, etc., are not covered.)

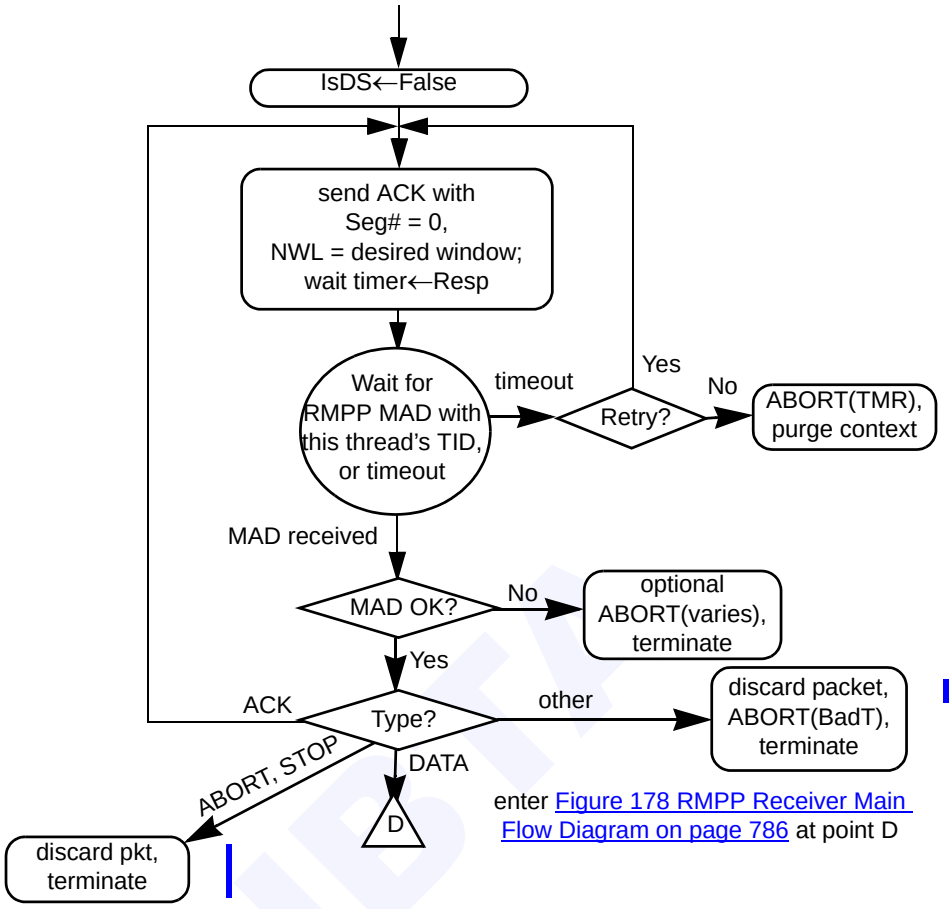


Figure 181 RMPP Sender Direction Switch Flow Diagram

1) Allocate a TID for the operation.

2) Allocate a receive context for the receiving operation, setting ES = 1 and IsDS=0.

3) Register the context with the dispatcher, using the TID, LID or GID of this endport (the Receiver), and the class of either the Sender or the Receiver depending on whether which is a class manager or agent.

4) Operating in that context, perform the actions shown in Figure 182 RMPP Receiver-Initiated Flow on page 792, ultimately either terminating or entering Figure 178 RMPP Receiver Main Flow Diagram on page 786 at point B.

On receiving the request MAD, the manager or agent on the Sender side should perform the steps described below in 13.6.6.2 Sender-Initiated Transfer on page 792, with the exception that instead of a new TID, it uses the TID provided in the request MAD. This will cause the first DATA packet to be sent (with RMPPFlags.Active=1), retried if needed. On receiving that first DATA packet, the Receiver-side dispatcher dispatches the previously prepared context and the transfer has begun.

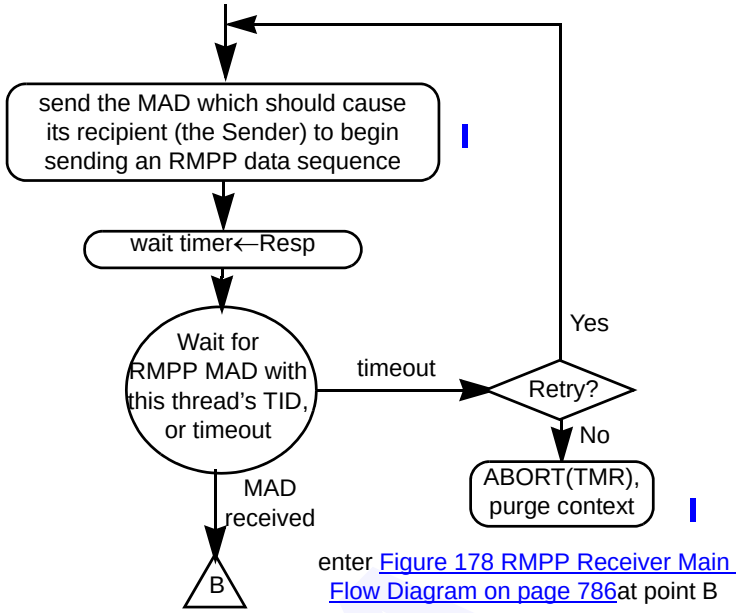


Figure 182 RMPP Receiver-Initiated Flow

13.6.6.2 SENDER-INITIATED TRANSFER

A Sender may initiate an RMPP transfer performing the following sequence (issues of allocating send buffers, posting send work requests, etc., are not covered):

1) Allocate a TID for the operation.
2) Allocate a send context for the receiving operation, setting the state variables as follows:
   - WF = WL = NS = 1; IsDS = 0.
   - other state variables indicating the class, method, attribute, data source, etc., of the packets to be sent as appropriate to the class and the implementation.
3) Register the context with the dispatcher, using the TID, the LID of the endnode this operation is being performed on, and the class of the manager performing this request or the target of the request packet.
4) Enter the Sender state diagram at the point labelled Send.

This will cause the request packet to be sent, and retries performed appropriately. The Receiver-side context is allocated and initiated on receipt of the first RMPP packet as indicated in Figure 176 RMPP Dispatcher on page 784.

13.6.6.3 SENDER-INITIATED DOUBLE-SIDED TRANSFER

It is also possible for a single transaction to involve an RMPP transfer sent in one direction followed by another RMPP transfer in the other direction,

typically sent in response. This is used, for example, in the GetMulti() method of SA. This may be accomplished as follows:

1) Allocate a TID, initialize and register with the dispatcher both send and receive contexts as described in the prior two sections, steps 1, 2, and 3, except that in the Sender context, IsDS=1.
2) Begin the initial transfer by starting the send operation at the point labelled Send. The method or other indication should be interpreted on the other side as initiating a double-sided transfer, causing the receive context to set IsDS=1.
3) When the Send operation terminates, IsDS=1 in the initially-used Sender and Receiver contexts will cause a transfer in the opposite direction to be initiated, according to the sequence described in 13.6.5.6 Direction Switch on page 790.

CHAPTER 14: SUBNET MANAGEMENT

14.1 SUBNET MANAGEMENT MODEL

Each subnet has at least one subnet manager (SM). Each SM resides on a port of a CA, router, or switch and can be implemented either in hardware or software. When there are multiple SMs on a subnet, one SM will be the master SM. The remaining SMs must be standby SMs. There is only one SM per port.

The master SM is a key element in initializing and configuring an IB subnet. The master SM is elected as part of the initialization process for the subnet and is responsible for:

- Discovering the physical topology of the subnet
- Assigning Local Identifiers (LIDs) to the endnodes, switches, and routers
- Establishing possible paths among the endnodes
- Sweeping the subnet, discovering topology changes and managing changes as nodes are added and deleted.

The communication between the master SM and the SMAs, and among the SMs, is performed with subnet management packets (SMPs). SMPs provide a fundamental mechanism for subnet management.

There are two types of SMPs: LID routed and directed route. LID routed SMPs are forwarded through the subnet (by the switches) based on the LID of the destination. Directed route SMPs are forwarded based on a vector of port numbers that define a path through the subnet. Directed route SMPs are used to implement several management functions, in particular, before the LIDs are assigned to the nodes. SMPs are specified in 14.2 Subnet Management Class on page 794.

Every switch, CA, and router has a subnet management agent (SMA), managed by the master SM. SMA are specified in 14.3 Subnet Management Agent on page 852.

The details of operation for both master and standby SMs are described in 14.4 Subnet Manager on page 859

14.2 SUBNET MANAGEMENT CLASS

This section defines the Subnet Management class of MADs. These MADs are also referred to as Subnet Management Packets, or SMPs. The purpose of this class is to provide subnet configuration, monitoring and

query of nodes within a subnet. SMPs are exchanged between a SM and SMAs on the subnet as described in Table 122 SM MAD Sources and Destinations on page 756.

There are two management classes dedicated to subnet management.

C14-1: The subnet management classes **shall** be identified by the MAD-Header:MgmtClass value of 0x01 for the LID Routed class and 0x81 for the Directed Route class as listed in Table 113 Management Class Values on page 720.

This section will describe class-specific methods, attributes, standard header fields and protocols for the Subnet Management Classes.

14.2.1 DATAGRAM FORMATS AND USE

C14-2: The datagrams in this class **shall** conform to the MAD format and usage rules as specified in 13.4.2 Management Datagram Format on page 718.

14.2.1.1 SMP DATA FORMAT - LID ROUTED

LID Routed SMPs are routed through the subnet using the normal switch forwarding tables set up during subnet initialization.

C14-3: A LID routed SMP **shall** have a format shown in Figure 183 on page 795 and Table 125 on page 796.

bytes	bits 31-24	bits 23-16	bits 15-8	bits 7-0
0	Common MAD Header			
...				
20				
24	M_Key			
28				
32	Reserved (32 bytes)			
...				
60				
64	SMP Data (64 bytes)			
128	Reserved (128 bytes)			
252				

Figure 183 SMP Format (LID Routed)

Table 125 SMP Fields (LID Routed)

Object	Length	Description
Common MAD Header	24 bytes	Common MAD as described in 13.4.2 Management Datagram Format on page 718.
M_Key	8 bytes	A 64 bit key, which is employed for SM authentication. Usage is defined in 14.2.4 Management Key on page 806.
Reserved	32 bytes	For aligning the SMP data field with the directed route SMP data field.
SMP Data	64 bytes	64 byte field of SMP data used to contain the method's attribute.
Reserved	128 bytes	Reserved.

14.2.1.2 SMP DATA FORMAT - DIRECTED ROUTE

Directed route SMPs are routed through the subnet from SMA to SMA using a store-and-forward technique between neighboring nodes. They are therefore not dependent on routing table entries. Directed route SMPs are primarily used for discovering the physical connectivity of a subnet before it has been initialized.

C14-4: A Directed Routed SMP **shall** have a format shown in Figure 184 SMP Format (Directed Route) on page 796 and Table 126 SMP Fields (Directed Route) on page 797.

bytes	bits 31-24	bits 23-16	bits 15-8	bits 7-0
0	Common MAD Header1			
4	D	Status	Hop Pointer	Hop Count
...	Common MAD Header2			
20				
24	M_Key			
28				
32	DrSLID		DrDLID	
36	Reserved (28 bytes)			
...				
60				
64	SMP Data (64 bytes)			
128	Initial Path (64 bytes)			

Figure 184 SMP Format (Directed Route)

bytes	bits 31-24	bits 23-16	bits 15-8	bits 7-0
192	Return Path (64 bytes)			
...				
252				

Figure 184 SMP Format (Directed Route) (Continued)

Table 126 SMP Fields (Directed Route)

Object	Length	Description
Common MAD Header1	4 bytes	Bytes 0-3 of the common MAD as described in 13.4.2 Management Datagram Format on page 718.
D	1 bit	Normally part of the class specific status field, this Direction bit is used by directed routing to determine direction of packet. If 0, the direction is outbound, from SM to node. If 1, the direction is inbound, from node to SM.
Status	15 bits	Code indicating status of method, as defined in 13.4.7 Status Field on page 731. There are no SMP status bits (bits 14-8 must be zero).
Hop Pointer	1 byte	Hop Pointer is used to indicate the current byte of the Initial/Return Path field.
Hop Count	1 byte	Hop Count is used to contain the number of valid bytes in the Initial/Return Path. It indicates how many direct route 'hops' to take.
Common MAD Header2	16 bytes	Bytes 8-23 of common MAD as described in 13.4.2 Management Datagram Format on page 718.
M_Key	8 bytes	A 64-bit key, which is employed for SM authentication. Usage is defined in 14.2.4 Management Key on page 806.
DrSLID	2 bytes	Directed route source LID. Used in directed routing.
DrDLID	2 bytes	Directed route destination LID. Used in directed routing.
Reserved	28 bytes	For the purpose of aligning the Data field on a 64 byte boundary.
Data	64 bytes	64-byte field of SMP data used to contain the method's attribute.
Initial Path	64 bytes	64-byte field containing the initial directed path. Each byte in this field represents a port.
Return Path	64 bytes	64-byte field containing the returning directed path. Each byte in this field represents a port.

14.2.2 SMPs AND DIRECTED ROUTE ALGORITHM

Directed route SMPs provide a mechanism for forwarding management packets throughout a configured, unconfigured, or partially configured subnet. This mechanism can be used to discover nodes in the subnet, perform diagnostics or verify link connectivity, bypassing the normal switch LID forwarding mechanism.

There are two components that support this mechanism in subnets: the Permissive destination address and directed routing. The Permissive destination address is defined in 4.1 Terminology And Concepts on page 142. When a node, including switches, receives a packet with this address it forwards it to its Subnet Management Interface. Directed routing permits the definition of an explicit route, based on intervening switch port numbers, that a packet is to traverse throughout the subnet.

Directed routing, in general, progresses much more slowly than normal switching. This is because each switch along the route has to perform some processing on every directed routed packet. Moreover, the IBA permits nodes to reserve a minimal amount of buffering for processing of SMPs. As a result, SMPs may be discarded in the subnet if the injection rate exceeds the buffering and processing capacity of the subnet and endnodes. Therefore, it is recommended that directed routing only be used where necessary.

The directed routing algorithm provides a method to use normal LID routing on either side of the directed route. No support is provided for more than one directed route. It is not possible to specify two or more directed routes with intervening LID routes. This is illustrated in Figure 185 Complete route using directed routing on page 799. The complete route between two nodes is made up of three parts, each of them potentially empty:

- From the source node to the source switch. This part uses LID routing, the source node and the source switch are identified by their LIDs. There may be other switches between them but this portion of the subnet has already been configured to allow LID routing.
- From the source switch to the destination switch. This part uses direct routing. The route is specified by stating the port number a packet must use to leave a switch. This portion of the subnet need not have been configured to allow LID routing.
- From the destination switch to the destination node. This part uses LID routing, the destination node and the destination switch are identified by their LIDs. There may be other switches between them but this portion of the subnet has already been configured to allow LID routing.

Since each part may be empty, there are eight combinations, although only four are really useful:

- All three parts are empty; this refers to the case when HopCount=0, DrSLID=0xFFFF and DrDLID=0xFFFF. This is used to loopback to oneself before a LID has been assigned. See Figure 186 Loopback using directed routing on page 799.

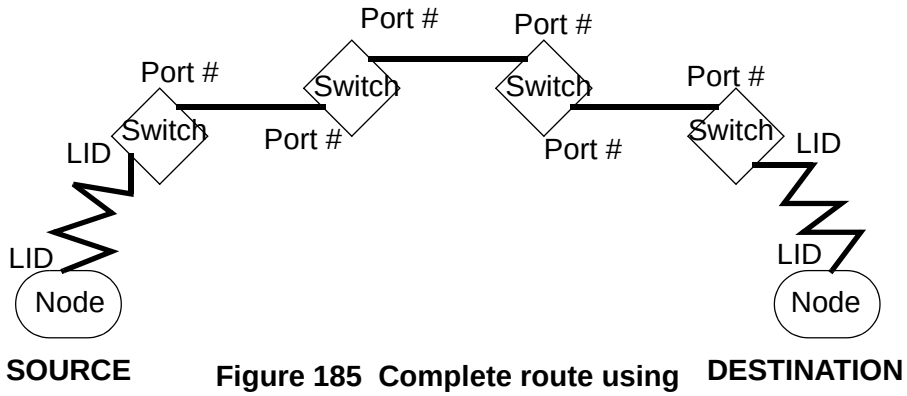


Figure 185 Complete route using directed routing

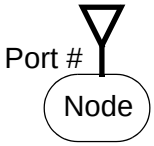


Figure 186 Loopback using directed routing

- Both LID routed parts are empty. This is a pure directed route used in a portion of a subnet not configured for LID routing. See Figure 187 Pure directed route on page 799.

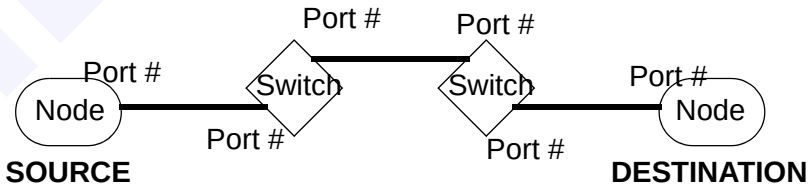


Figure 187 Pure directed route

- One of the LID routed part is empty. This is used when a portion of the subnet, either at the source or at the destination, has been configured for LID routing. See Figure 188 Directed route with LID-Routed part at the source on page 800 and Figure 189 Directed route with LID-Routed part at the destination on page 800.

- No part is empty. This is the general case as illustrated in Figure 185 Complete route using directed routing on page 799 when portions of the subnet have been initialized both at the source and destination but not in between.

The following section describes how a directed route packet is initialized, how a return packet is initialized, and the algorithm used along the route by switches to forward the packets. Note that in the switched portions of

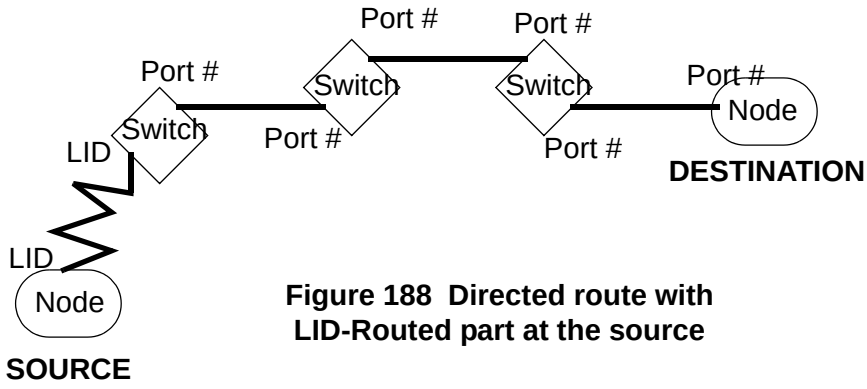


Figure 188 Directed route with LID-Routed part at the source

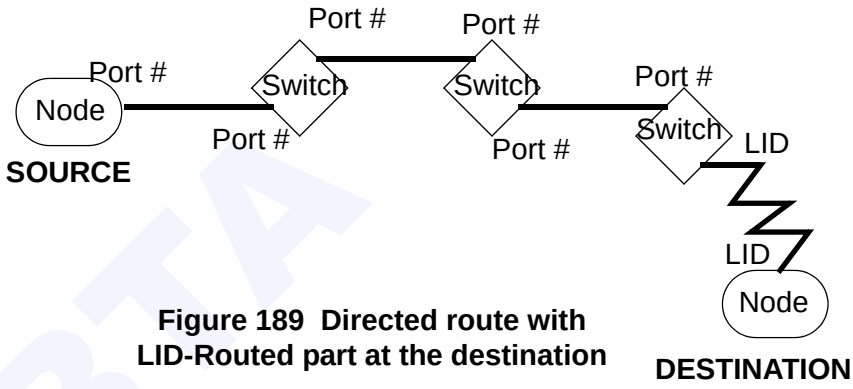


Figure 189 Directed route with LID-Routed part at the destination

a route, the nodes are not directly involved; the packet is switched along the path just as any other packet is.

14.2.2.1 OUTGOING DIRECTED ROUTE SMP INITIALIZATION

C14-5: Only a SM **shall** originate a directed route SMP.

C14-5.a1: The SM **shall** insure that all nodes on a LID routed segment **shall** be properly initialized for LID routing.

In the following sections, the source node where the SM originator resides is called the requester node, and the destination node is called the responder node, even when describing the return process. When directed route is used, it refers to the directed route part only, not the complete route.

C14-6: The fields of the directed route SMP **shall** be initialized as follows:

- Mgmt Class **shall** be set to the directed route Subnet Management class as specified in Table 113 Management Class Values on page 720.
- Method **shall** be set to SubnGet() or SubnSet() as specified in Table 114 Common Management Methods on page 722.

- D bit **shall** be set to 0.
- Hop Pointer **shall** be set to 0.
- Hop Count **shall** be set to the number of links traversed along the directed route part of the path. Valid values are from 0 to 63. In Figure 185 Complete route using directed routing on page 799, this number would be 3. In Figure 186 Loopback using directed routing on page 799, this number would be 0.
- If the directed route part starts from the requester node, i.e. there is no LID routed part at the source as illustrated in Figure 186 Loopback using directed routing on page 799, Figure 187 Pure directed route on page 799 or Figure 189 Directed route with LID-Routed part at the destination on page 800, then the DrSLID **shall** be set to the Permissive LID. If the directed route does *not* start from the requester node, then DrSLID **shall** be set to the LID of the requester node, which must have been assigned.
- If the directed route ends at the responder node, i.e. there is no LID routed part at the destination as illustrated in Figure 186 Loopback using directed routing on page 799, Figure 187 Pure directed route on page 799 or Figure 188 Directed route with LID-Routed part at the source on page 800, then the DrDLID **shall** be set to the Permissive LID. If the directed route does *not* end at the responder node, then the DrDLID **shall** be set to the LID of the responder node, which must have been assigned.
- Initial Path **shall** be set to an array of Hop Count port numbers corresponding to the ports at the starting end of hops, specifically, the port from which the SMP will start travelling on the inter-node link along the directed route. The array **shall** be laid out in the initial Path field so that the byte at offset 0 in that field is reserved and the following bytes are filled with the port numbers in order.
- Return Path **shall** be set to an array of Hop Count zeroes. The array **shall** be laid out in the Return Path field so that the byte at offset 0 in that field is reserved and the following bytes are filled with zeroes.
- All other fields **shall** be set the same way they are for a LID routed SMP.

C14-7: The data packet headers for the unreliable datagram encapsulating the directed route SMP **shall** be initialized as follows:

- If the directed route part starts from the requester node, the LRH:SLID **shall** be set to the Permissive LID or a LID of this port. If the directed route does *not* start from the requester node, the LRH:SLID **shall** be set to the LID of the requester node, which must have been assigned.

- LRH:DLID **shall** be set to the Permissive LID if the directed route part starts from the requester node. If not, it **shall** be set to the LID of the source switch in the directed route part. That LID must have been assigned and routing must have been initialized between that switch and the requester node.
- All other fields **shall** be set the same way they are for a LID routed SMP.

The SM will then hand the packet to the SMI. If the directed route part starts from the requester node, the SMI processes the packet as described in 14.2.2.2 Outgoing Directed Route SMP handling by SMI on page 802. Otherwise, the SMI will output the packet as it does any LID routed packet.

14.2.2.2 OUTGOING DIRECTED ROUTE SMP HANDLING BY SMI

C14-8: Any SMP arriving at the SMI with a MADHeader:MgmtClass set to 0x81 (Directed Route class) **shall** be processed by the SMI.

C14-9: The SMI **shall** handle outgoing directed route SMPs (D bit is 0) as defined by one of the following mutually exclusive cases:

1) If Hop Count is nonzero and Hop Pointer is zero, the SMI is at the beginning of the directed route portion of the path. The SMI **shall** alter the contents of the directed route SMP as follows:
   - The Hop Pointer **shall** be incremented by 1.
   - For switches, the LRH:SLID **shall** be set to the Permissive LID. For CA's the LRH:SLID **shall** be set to a LID of this port or the Permissive LID.
   - LRH:DLID **shall** be set to the Permissive LID.

   The SMI **shall** output the packet on the port whose number is in the entry indexed by Hop Pointer in the Initial Path. If that port number is invalid, the SMI **shall** discard the SMP.
2) If Hop Count is non zero and Hop Pointer is in the range 1 ≤ Hop Pointer < Hop Count, this SMI is an intermediate hop in the directed route portion of the path. If the node is not a switch, the SMI **shall** discard the SMP, otherwise the SMI **shall** alter the contents of the directed route SMP as follows:
   - The entry indexed by Hop Pointer in the Return Path array of port numbers **shall** be set to the port number where the SMP was received and then the Hop Pointer **shall** be incremented by 1.
   - The LRH:SLID **shall** be set to the Permissive LID.
   - The LRH:DLID **shall** be set to the Permissive LID.

The SMI **shall** output the packet on the port whose number is in the entry indexed by Hop Pointer in the Initial Path. If that port number is invalid, the SMI **shall** discard the SMP.

3) If Hop Pointer is equal to Hop Count, the SMI is at the end of the directed route portion of the path. The SMI **shall** alter the contents of the directed route SMP as follows:

   - If the Hop Count is zero, the Hop Pointer **shall** be incremented by 1. Otherwise, the entry indexed by Hop Pointer in the Return Path array of port numbers **shall** be set to the port number where the SMP was received and then the Hop Pointer **shall** be incremented by 1.
   - For switches, the LRH:SLID of the outgoing directed route SMP **shall** be altered as follows: If the DrDLID is the Permissive LID, the LRH:SLID **shall** be set to the Permissive LID. If the DrDLID is *not* the Permissive LID, the LRH:SLID **shall** be set to the base LID of this node. For CA's, if the DrDLID is the Permissive LID, the LRH:SLID may be set to the Permissive LID; otherwise, the SMP is silently dropped.
   - The LRH:DLID **shall** be set to the DrDLID.

   If the LRH:DLID is the Permissive LID, this node is the responder node and the SMI **shall** hand the packet to the SMA or SM, which may check that Hop Pointer is equal to HopCount+1.

   If the LRH:DLID is *not* the Permissive LID, the SMI will output the packet as it does any LID routed packet.

4) If Hop Pointer is equal to Hop Count+1, this node is the responder node and the SMI **shall** hand the packet to the SMA or SM, which may check that Hop Pointer is equal to Hop Count+1.

5) If Hop Pointer is in the range (Hop Count+1) < Hop Pointer ≤ 255, the SMI **shall** silently discard the SMP.

The handling of returning directed route SMPs (D bit is 1) is described in 14.2.2.3 Returning Directed Route SMP Initialization on page 803.

14.2.2.3 RETURNING DIRECTED ROUTE SMP INITIALIZATION

The SMA or SM receiving a directed route SMP processes it (with regard to handling of the method and attribute) as it does a LID routed SMP. The receiving SMA or SM may determine that it should send a response.

C14-10: The fields of the directed route response SMP **shall** be initialized as follows:

- Method **shall** be set to SubnGetResp() as specified in Table 114 Common Management Methods on page 722.
- D bit **shall** be set to 1.

- Mgmt Class, Hop Pointer, Hop Count, DrSLID, DrDLID, Initial Path and Return Path **shall** be copied as is from the request SMP.
- All other fields **shall** be set the way they are set for a LID routed SMP.

C14-11: The data packet headers for the unreliable datagram encapsulating the directed route response SMP **shall** also be initialized as follows:

- If the directed route part starts from the responder node, the LRH:SLID **shall** be set to the Permissive LID or a LID of this port. If the directed route does not start from the responder node, the LRH:SLID **shall** be set to the LID of the responder node, which must have been assigned.
- The LRH:DLID **shall** be set to the LRH:SLID of the directed route request SMP.
- All other fields **shall** be set the way they are set for a LID routed SMP.

The SMA or SM will then hand the packet to the SMI. If the directed route part starts from the responder node, the SMI processes the packet as described in 14.2.2.4 Returning Directed Route SMP handling by SMI on page 804. Otherwise, the SMI will output the packet as it does any LID routed packet.

14.2.2.4 RETURNING DIRECTED ROUTE SMP HANDLING BY SMI

C14-12: This compliance statement is obsolete and has been removed.

C14-13: The SMI **shall** handle returning directed route SMPs (D bit is 1) as defined by one of the following mutually exclusive cases:

1) If Hop Count is non-zero and Hop Pointer is equal to Hop Count + 1, the SMI is at the beginning of the directed route portion of the path. The SMI **shall** alter the contents of the directed route SMP as follows:
   - Hop Pointer **shall** be decremented by 1.
   - For switches, the LRH:SLID **shall** be set to the Permissive LID. For CA's the LRH:SLID **shall** be set to a LID of this port or the Permissive LID.
   - LRH:DLID **shall** be set to the Permissive LID.

   The SMI **shall** output the packet on the port whose number is in the entry indexed by Hop Pointer in the Return Path. If that port number is invalid, the SMI **shall** discard the SMP.
2) If Hop Count is non-zero and Hop Pointer is in the range $2 \leq$ Hop Pointer $\leq$ HopCount, this SMI is an intermediate hop in the directed route portion of the path. If the node is not a switch, the SMI **shall** discard the SMP, otherwise, he SMI **shall** alter the contents of the directed route SMP as follows:

- Hop Pointer **shall** be decremented by 1.
- The LRH:SLID **shall** be set to the Permissive LID.
- The LRH:DLID **shall** be set to the Permissive LID.

The SMI **shall** output the packet on the port whose number is in the entry indexed by Hop Pointer in the Return Path. If that port number is invalid, the SMI **shall** discard the SMP.

3) If Hop Pointer is equal to 1, the SMI is at the end of the directed route portion of the path. The SMI **shall** alter the fields of the directed route SMP as follows:

   - HopPointer **shall** be decremented by 1.
   - For switches, the LRH:SLID of the returning directed route SMP **shall** be altered as follows: If the DrSLID is the Permissive LID, the LRH:SLID **shall** be set to the Permissive LID. If the DrSLID is not the Permissive LID, the LRH:SLID **shall** be set to the LID of this node. For CA's, if the DrSLID is the Permissive LID, the LRH:SLID may be set to the Permissive LID; otherwise, the SMP **shall** be silently dropped.
   - LRH:DLID **shall** be set to the DrSLID.

   If the LRH:DLID is the Permissive LID, then this node is the requester node and the SMI must hand the packet to the SM which may check that HopPointer is equal to 0.

   If the LRH:DLID is *not* the Permissive LID, the SMI will output the packet as it does any LID routed packet.

4) If Hop Pointer is equal to 0, this node is the requester node and the SMI must hand the packet to the SM, which may check that Hop Pointer is equal to 0.

5) If Hop Pointer is in the range (Hop Count + 2) ≤ Hop Pointer ≤ 255, then the SMI **shall** silently discard the SMP.

The handling of outgoing directed route SMPs (D bit is 0) is described in 14.2.2.2 Outgoing Directed Route SMP handling by SMI on page 802.

14.2.3 METHODS

The Subnet Management class uses a subset of the common methods described in 13.4.5 Management Class Methods on page 721.

C14-13.1.1: Subnet management entities **shall** support the methods listed in Table 127 Subnet Management Methods on page 806. All method type values not listed in the table are reserved.

Table 127 Subnet Management Methods

Method Type	Value	Description
SubnGet()	0x01	Request a get (read) of an attribute.
SubnSet()	0x02	Request a set (write) of an attribute.
SubnGetResp()	0x81	Response from a get or set request.
SubnTrap()	0x05	Notify an event occurred.
SubnTrapRepress()	0x07	Cease sending repeated Trap.

Table 122 SM MAD Sources and Destinations on page 756 indicates which methods are applied to SMPs that originate at a SM, SMPs that originate at a SMA, and SMPs that may be destined to SMAs or to SMs.

C14-14: This compliance statement is obsolete and has been removed.

Subnet Management entities, the SMA and SM, support the methods listed in Table 127 Subnet Management Methods on page 806.

14.2.4 MANAGEMENT KEY

SMPs are used to initialize and configure CAs, switches and routers, and are therefore considered privileged operations. As a result, there is a mechanism provided to authorize subnet management operations based on:

- a Key stored in the MADHeader:M_Key of the LID routed and Directed route subnet management class datagram as shown in Figure 183 SMP Format (LID Routed) on page 795 and Figure 184 SMP Format (Directed Route) on page 796, respectively.
- a Key kept locally on each port in the PortInfo:M_Key component of the PortInfo attribute that is described in Table 145 PortInfo on page 822.

Authentication is performed by the management entity at the destination port and is achieved by comparing the key contained in the SMP with the key residing at the destination port. This key is known as the Management Key (M_Key).

C14-15: An M_Key contained in the MADHeader:M_Key of the SMP **shall** not be checked at the receiving port with the PortInfo:M_Key set to zero. As a result, no authentication is performed.

If the PortInfo:M_Key is nonzero, authentication at the receiving port and access to the port attributes is determined by the contents of the PortInfo:M_KeyProtectBits as described in 14.2.4.1 Levels of Protection on page 807. Finally, M_Keys can be lost, so Key recovery is provided by the PortInfo:M_KeyLeasePeriod components and is described in 14.2.4.2 Lease Period on page 807.

Note that M_Key components of PortInfo are undefined for switch physical ports. M_Key checking for switch physical port attributes is performed using PortInfo:M_Key of switch port 0.

14.2.4.1 LEVELS OF PROTECTION

C14-16: If the PortInfo:M_Key is non-zero, the management entity residing at the port **shall** perform authentication determined by the contents of the PortInfo:M_KeyProtectBits and the behaviors described in Table 128 Protection Levels on page 807.

Table 128 Protection Levels

PortInfo:M_KeyProtectBits	Description
0	SubnGet(*) **shall** succeed for any key in the MADHeader:M_Key and SubnGetResp(PortInfo) **shall** return the contents of the PortInfo:M_Key component. SubnSet(*) and SubnTrapRepress(*) **shall** fail if MADHeader:M_Key does not match the PortInfo:M_Key component in the port.
1	SubnGet(*) **shall** succeed for any key in the MADHeader:M_Key and SubnGetResp(PortInfo) **shall** return the contents of the PortInfo:M_Key component set to zero if MADHeader:M_Key does not match the PortInfo:M_Key component in the port. SubnSet(*) and SubnTrapRepress(*) **shall** fail if MADHeader:M_Key does not match the PortInfo:M_Key component in the port.
2	SubnGet(*), SubnSet(*), and SubnTrapRepress(*) **shall** fail if MADHeader:M_Key does not match the PortInfo:M_Key component in the port.
3	SubnGet(*), SubnSet(*), and SubnTrapRepress(*) **shall** fail if MADHeader:M_Key does not match the PortInfo:M_Key component in the port. This function is identical to the function of PortInfo:M_KeyProtectBits = 2. It is retained only for backward compatibility.

14.2.4.2 LEASE PERIOD

A Lease Period is specified by setting the contents of the PortInfo:M_KeyLeasePeriod component. It is intended to allow an M_Key to 'expire' if the master SM inadvertently goes away without sharing the

M_Key with backup SMs and there is no other out-of-band recovery mechanism available.

C14-17: The lease period timer **shall** start counting down toward zero on a port when a SMP is received for which the M_Key check was performed according to Table 128 Protection Levels on page 807 and failed. If the lease timer countdown is already underway, it **shall** not be interrupted by the arrival of that SMP.

C14-18: The PortInfo:M_KeyViolations component **shall** be incremented on a port when a SMP is received by that port for which the M_Key check was performed according to Table 128 Protection Levels on page 807 and failed. The incrementing **shall** stop when the component reaches all 1s.

Furthermore, an M_Key violation trap/notice must be generated as described in 14.3.9 M_Key mismatch on page 857 indicating that the lease timer has started counting. In response to that trap, the master SM may refresh the Lease Period. If the master SM that originally set the M_Key has gone away, the Lease Period may expire.

C14-19: The lease period counter **shall** cease counting down and **shall** be reset to the value contained in PortInfo:M_KeyLeasePeriod component on a port when any SMP is received with MADHeader:M_Key that matches the PortInfo:M_Key.

C14-20: The PortInfo:M_KeyProtectBits **shall** be set to zero when the lease period counter transitions from non-zero to zero.

When the lease period expires, clearing the M_Key Protection bits will allow any SM to read (and then set) the M_Key.

C14-21: When the PortInfo:M_KeyLeasePeriod is set to zero, the lease period **shall** never expire.

Whether there is an out-of-band mechanism to reset data protected with a lease period of zero is outside the scope of the specification.

14.2.4.3 NOTES ON EXPECTED USAGE

- The SM is responsible for keeping track of the M_Keys for the nodes that it is managing, to make sure that it uses the correct key for each node.
- If standby SMs exist in the subnet for redundancy, then the M_Keys may be shared so that failover to another SM can be accommodated easily.
- An SM may have exclusive access to a node (or set of nodes), by using an M_Key which is only known by that SM and the particular node(s).

- SubnSet() is always protected by this mechanism as it can affect the state of the node. SubnGet() is protected only if PortInfo:M_KeyProtectBits is appropriately set.

14.2.4.4 UPDATE PROCEDURE

Node protection/ownership is assigned in one “atomic” operation.

C14-22: The PortInfo:M_Key, the PortInfo:M_KeyProtectBits, the PortInfo:M_KeyLeasePeriod components in the PortInfo Attribute **shall** be set in one SubnSet(PortInfo) method.

A returned SubnGetResp(PortInfo) with a status of zero indicates to the SM that it has taken ownership of the node.

14.2.4.5 INITIALIZATION

C14-23: When initially powered-up or reset, the PortInfo:M_Key, the PortInfo:M_KeyProtectBits, the PortInfo:M_KeyLeasePeriod components of an endport **shall** be set to zero if NVRAM is not used or to a value stored in NVRAM.

If the M_Key related components are not stored in NVRAM, the PortInfo:M_Key, the PortInfo:M_KeyProtectBits, the PortInfo:M_KeyLeasePeriod components may be set by any master SM during subnet initialization. Initialization of M_KeyLeasePeriod to a value of zero (infinite) notwithstanding, whenever a port's M_Key-related components are not stored in NVRAM, any subnet manager can successfully read and then set the port’s M_Key during subnet initialization.

14.2.4.6 SMI

The SMI will not check the M_Key in the header of a SMP since that is the responsibility of the management entities that reside behind the SMI.

14.2.5 ATTRIBUTES

In the SMP, attributes can be up to 64 bytes long. Table 129 Subnet Management Attributes (Summary) on page 810 summarizes the subnet management attributes and Table 130 Subnet Management Attribute / Method Map on page 811 indicates which methods apply to each attribute.

C14-24: This compliance statement is obsolete and has been replaced by C14-24.1.1:

C14-24.1.1: Subnet management entities **shall** support the attributes and methods as listed in Table 129 Subnet Management Attributes (Summary) on page 810 and Table 130 Subnet Management Attribute / Method Map

on page 811. All attribute IDs not listed in Table 129: Subnet Management Attributes (Summary) are reserved.

Table 129 Subnet Management Attributes (Summary)

Attribute Name	Attribute ID	AttributeModifier	Description	Applicable To
Notice	0x0002	0x0000_0000	Information regarding the associated Notice or Trap(). See 14.2.5.1 Notices and Traps on page 812.	All Endports on All Nodes
NodeDescription	0x0010	0x0000_0000	Node Description String. See 14.2.5.2 NodeDescription on page 818.	All Nodes
NodeInfo	0x0011	0x0000_0000	Generic Node Data. See 14.2.5.3 NodeInfo on page 818.	All Ports on All Nodes
SwitchInfo	0x0012	0x0000_0000	Switch Information. See 14.2.5.4 SwitchInfo on page 819.	Switches
GUIDInfo	0x0014	GUID Block	Assigned GUIDs. See 14.2.5.5 GUIDInfo on page 821.	All Endports
PortInfo	0x0015	Port Number	Port Information. See 14.2.5.6 PortInfo on page 821.	All Ports on All Nodes
P_KeyTable	0x0016	Port Number/P_Key block	Partition Table. See 14.2.5.7 P_KeyTable on page 834.	All Ports on All Nodes
SLtoVLMappingTable	0x0017	Input/Output Port Number	Service Level to Virtual Lane mapping Information. See 14.2.5.8 SLtoVLMappingTable on page 835.	All Ports on All Nodes (optional[a])
VLArbitrationTable	0x0018	Output Port/Component	List of Weights. See 14.2.5.9 VLArbitrationTable on page 836.	All Ports on All Nodes (optional[b])
LinearForwardingTable	0x0019	Block Identifier	Linear Forwarding Table Information. See 14.2.5.10 LinearForwardingTable on page 837.	Switches (optional[c])
RandomForwardingTable	0x001A	Block Identifier	Random Forwarding Table Information. See 14.2.5.11 RandomForwardingTable on page 838.	Switches (optional[c])
MulticastForwardingTable	0x001B	Block Identifier	Multicast Forwarding Table Information. See 14.2.5.12 MulticastForwardingTable on page 838.	Switches (optional)
SMInfo	0x0020	0x0000_0000 - 0x0000_0005	Subnet Management Information. See 14.2.5.13 SMInfo on page 840.	All nodes hosting an SM

Table 129 Subnet Management Attributes (Summary) (Continued)

Attribute Name	Attribute ID	AttributeModifier	Description	Applicable To
VendorDiag	0x0030	0x0000_0000 - 0x0000_FFFF	Vendor Specific Diagnostic. See 14.2.5.14 VendorDiag on page 840.	All Ports on All Nodes
LedInfo	0x0031	0x0000_0000	Turn on/off LED. See 14.2.5.15 LedInfo on page 842.	All nodes (optional)
	0xFF00-0xFFFF	0x0000_0000 - 0xFFFF_FFFF	Range reserved for Vendor Specific attributes.	

a. Optional on ports that support only one data VL.
b. Prohibited on ports that support only one data VL.
c. LinearForwardingTable and RandomForwardingTable are mutually exclusive, but one is required.

Table 130 Subnet Management Attribute / Method Map

Attribute Name	Get	Set	Trap	TrapRepress
Notice	X	X	X	X
NodeDescription	X			
NodeInfo	X			
SwitchInfo	X	X		
GUIDInfo	X	X		
PortInfo	X	X		
P_KeyTable	X	X		
SLtoVLMappingTable	X	X		
VLArbitrationTable	X	X		
LinearForwardingTable	X	X		
RandomForwardingTable	X	X		
MulticastForwardingTable	X	X		
SMInfo	X	X		
VendorDiag	X			
LedInfo	X	X		

Several of the SM attributes described in the sections that follow (LinearForwardingTable, RandomForwardingTable,MulticastForwardingTable, VLArbitrationTable, GUIDInfo, and P_KeyTable) are used to load and read contents of tables in switches and CAs. Each of those attributes uses a block of table entries and an offset into the table specified using the

MadHeader:AttributeModifier component. Successive block elements are loaded or read starting at the specified offset.

The number of table entries in a block is fixed, and the offset (AttributeModifier) is specified in units of block size. Also, there is no requirement that the actual table lengths, as indicated by related Cap or Top components, be an integer multiple of the block size.[23]

As long as some of a block's entries address valid table entries, this is not an error. The block elements corresponding to invalid table entries are ignored on write and read back as all zeros.

However, if none of the block entries address valid table entries, it must mean that the AttributeModifer value specifies an offset that is past the end (Cap or Top) of the table. In this case a MADHeader:Status.Code of 7 is returned, since that is the status code for an invalid attribute modifier value (see 13.4.7 Status Field on page 731). The attribute contents accompanying that Status Code are implementation-specific.

14.2.5.1 NOTICES AND TRAPS

This attribute is a common attribute described in 13.4.8.2 Notice on page 737. The following traps are defined for the Subnet Management class.

Table 131 Traps

Trap Number	Type[a]	Kind of Sending Node	DataDetails
64	Informational	subnet manager	<GIDADDR> is now in service; see 14.4.9 In and Out of Service Traps on page 880
65	Informational	subnet manager	<GIDADDR> is out of service; see 14.4.9 In and Out of Service Traps on page 880
66	Informational	subnet manager	New multicast group with multicast address <GIDADDR> is now created; see 14.4.10 Multicast Group Create/Delete Traps on page 880
67	Informational	subnet manager	Multicast group with multicast address <GIDADDR> is now deleted; see 14.4.10 Multicast Group Create/Delete Traps on page 880
128	Urgent	switch	Link state of at least one port of switch at <LIDADDR> has changed. ; see 14.3.6 Port State Change on page 855.

23. All of the table attributes except LinearForwardingTable use a Cap component only. For example, SwitchInfo:RandomFDBCap is used for RandomForwardingTable. The LinearForwardingTable uses both a Cap and a Top. See 14.2.5.10 LinearForwardingTable on page 837. This implies that the last, i.e., largest offset, block that is loaded or read from a table may overflow past the end of a table, implicitly addressing table entries that are invalid. When the table size happens to be less than the block size, such a "last" block will also be the first, and only valid, block.

Table 131 Traps (Continued)

Trap Number	Type[a]	Kind of Sending Node	DataDetails
129	Urgent	any	Local Link Integrity threshold reached at <LIDADDR><PORTNO>
130	Urgent	any	Excessive Buffer Overrun threshold reached at <LIDADDR><PORTNO>
131	Urgent	switch	Flow Control Update watchdog timer expired at <LIDADDR><PORTNO>
144	Informational	any	The CapabilityMask at <LIDADDR> has been modified and its new value is <CAPABILITYMASK>. This trap is optional; see 14.3.11 Change CapabilityMask on page 858.
145	Informational	any	The SystemImageGUID at <LIDADDR> has been modified and its new value is <SYSTEMIMAGEGUID>. This trap is optional; see 14.3.12 Change SystemImageGUID on page 858
256	Security	any	Bad M_Key, <MKEY> from <LIDADDR> attempted <METHOD> with <ATTRIBUTEID> and <ATTRIBUTEMODIFIER>; if MAD was directed route, see C14-24.1.2: on page 813 and C14-24.1.3: on page 813
257	Security	any	Bad P_Key, <KEY> from <LIDADDR1>/<GIDADDR1>/<QP1> to <LIDADDR2>/<GIDADDR2>/<QP2> on <SL>.
258	Security	any	Bad Q_Key, <KEY> from <LIDADDR1>/<GIDADDR1>/<QP1> to <LIDADDR2>/<GIDADDR2>/<QP2> on <SL>.
259	Security	switch	Bad P_Key <PKEY> from <LIDADDR1>/<GIDADDR1>/<QP1> to <LIDADDR2>/<GIDADDR2>/<QP2> on <SL> at switch <LIDADDR> external port <PORTNO>, where the validity of all the above fields is indicated by <DataValid>. This trap is optional; see 14.3.7 P_Key Mismatch on Switch External Ports on page 856.

a. For these traps, the type field is ignored in all MADs using the Notice attribute.

Traps use the layouts shown in the tables below for the DataDetails component of their Notice attributes.

C14-24.1.2: Unless the Directed Route Notice option is implemented by an SMA as indicated by PortInfo:CapabilityMask.IsDRNoticeSupported, all components of all SMP Notice DataDetails for Trap 256 whose name begins with the letters “DR” **shall** be ignored on read; their content is unspecified.

C14-24.1.3: If the Directed Route Notice option is implemented by an SMA (PortInfo:CapabilityMask.IsDRNoticeSupported=1), and an M_Key violation is caused as a result of the arrival of a LID Routed SMP, the SMA **shall** set to 0 the DRNotice attribute of the associated Notice's DataDetails. If the M_Key violation was caused by the arrival of a Directed Route SMP, the SMA **shall** set the following notice attributes in the associated Notice's DataDetails as specified below:

- DRNotice **shall** be set to 1.

- DRPathTruncated **shall** be set to 0 if the return path of the directed route SMP is short enough to fit completely in the Notice's DRNoticeReturnPath component; otherwise DRPathTruncated **shall** be set to 1.
- DRHopCount **shall** be set to the number of hops placed in this Notice.
- DRSLID **shall** be set to the DrSLID component of the Directed Route SMP.
- The ReturnPath of the SMP **shall** be placed in the DRNoticeReturnPath field. The lowest-offset ReturnPath byte **shall** be placed in the lowest offset DRNoticeReturnPath byte; the second-lowest in the second-lowest; etc. If the ReturnPath is too long to fit in the Notice's DRNoticeReturnPath, the highest offset bytes of the ReturnPath are not recorded.

Generation of this Notice takes place in the SMA, after the Directed Route SMP has been processed by the SMI.

Table 132 Notice DataDetails For Traps 64, 65, 66, and 67

Field	Length(bits)	Description
Reserved	48	Reserved
GIDADDR	128	Global Identifier
Padding	256	**Shall** be ignored on read. Content is unspecified.

Table 133 Notice DataDetails For Trap 128

Field	Length(bits)	Description
LIDADDR	16	Local Identifier
Padding	416	**Shall** be ignored on read. Content is unspecified.

Table 134 Notice DataDetails For Traps 129, 130 and 131

Field	Length(bits)	Description
Reserved	16	Reserved
LIDADDR	16	Local Identifier
PORTNO	8	Port number
Padding	392	**Shall** be ignored on read. Content is unspecified.

Table 135 Notice DataDetails For Trap 144

Field	Length(bits)	Description
Reserved	16	Reserved
LIDADDR	16	Local Identifier
Reserved	16	Reserved
CAPABILITYMASK	32	Contents of the CapabilityMask at <LIDADDR>
Padding	352	**Shall** be ignored on read. Content is unspecified.

Table 136 Notice DataDetails For Trap 145

Field	Length(bits)	Description
Reserved	16	Reserved
LIDADDR	16	Local Identifier
Reserved	16	Reserved
SYSTEMIMAGEGUID	64	Contents of the SystemImageGUID at <LIDADDR>
Padding	320	**Shall** be ignored on read. Content is unspecified.

Table 137 Notice DataDetails For Trap 256

Field	Length(bits)	Description
Reserved	16	Reserved
LIDADDR	16	Local Identifier
DRSLID	16	DrSLID of the directed route SMP causing this Notice, if any. See C14-24.1.3: on page 813
METHOD	8	Method
Reserved	8	Reserved
ATTRIBUTEID	16	Attribute ID
ATTRIBUTEMODIFIER	32	AttributeModifier
MKEY	64	M_Key
Reserved	8	Reserved
DRNotice	1	Indicates whether this Notice results from a directed route SMP See C14-24.1.3: on page 813.

Table 137 Notice DataDetails For Trap 256 (Continued)

Field	Length(bits)	Description
DRPathTruncated	1	Indicates that the DRNoticeReturnPath is truncated. See C14-24.1.3: on page 813.
DRHopCount	6	Number of bytes in the DRNoticeReturnPath. See C14-24.1.3: on page 813.
DRNoticeReturnPath	240	ReturnPath from the directed route SMP. See C14-24.1.3: on page 813.

Table 138 Notice DataDetails For Traps 257 and 258

Field	Length(bits)	Description
Reserved	16	Reserved
LIDADDR1	16	Local Identifier
LIDADDR2	16	Local Identifier
KEY	32	Q_Key or P_Key. If P_Key, the 16 most significant bits of the field **shall** be set to 0 and the 16 least significant bits of the field **shall** be set to the P_Key.
SL	4	Service Level
Reserved	4	Reserved
QP1	24	Queue Pair
Reserved	8	Reserved
QP2	24	Queue Pair
GIDADDR1	128	Global Identifier. If no GRH is present in the offending packet, this field **shall** be filled with zeroes.
GIDADDR2	128	Global Identifier. If no GRH is present in the offending packet, this field **shall** be filled with zeroes.
Padding	32	**Shall** be ignored on read. Content is unspecified.

Table 139 Notice DataDetails For Trap 259

Field	Length(bits)	Description
DataValid	16	Indicates validity of optional data fields; 0 = field invalid; 1 = field valid: • bit 0: LIDADDR1 • bit 1: LIDADDR2 • bit 2: PKEY • bit 3: SL • bit 4: QP1 • bit 5: QP2 • bit 6: GIDADDR1 • bit 7: GIDADDR2 • bits 8-15: Reserved
LIDADDR1	16	Local Identifier
LIDADDR2	16	Local Identifier
PKEY	16	P_Key
SL	4	Service Level
Reserved	4	Reserved
QP1	24	Queue Pair
Reserved	8	Reserved
QP2	24	Queue Pair
GIDADDR1	128	Global Identifier. Shall be set to zero to indicate lack of a GRH in the offending packet when Data-Valid bit 6 is 1.
GIDADDR2	128	Global Identifier. Shall be set to zero to indicate lack of a GRH in the offending packet when Data-Valid bit 7 is 1.
SWLIDADDR	16	Local Identifier of switch
PORTNO	8	Port Number
Padding	24	**Shall** be ignored on read. Content is unspecified.

14.2.5.2 NODEDESCRIPTION

Table 140 NodeDescription

Component	Access	Length(bits)	Description
NodeString	RO	512	UTF-8 encoded string to describe node in text format.

The contents of the NodeDescription attribute are the same for all ports on a node.

14.2.5.3 NODEINFO

The NodeInfo Attribute provides fundamental management information common to all CAs, routers, and switches. It **shall** be implemented by all nodes.The value of some NodeInfo components varies by port within a node.

Table 141 NodeInfo

Component	Access	Length (bits)	Offset (bits)	Description
BaseVersion[a]	RO	8	0	Supported MAD Base Version. Indicates that this node supports up to and including this version. Set to 1.
ClassVersion[a]	RO	8	8	Supported Subnet Management Class (SMP) Version. Indicates that this node supports up to and including this version. Set to 1.
NodeType[a]	RO	8	16	1: Channel Adapter 2: Switch 3: Router 0, 4 - 255: Reserved
NumPorts[a]	RO	8	24	Number of physical ports on this node.
SystemImageGUID[a]	RO	64	32	GUID associating this node with other nodes controlled by common supervisory code. Provides a means for system software to indicate the availability of multiple paths to the same destination via multiple nodes. Set to zero if indication of node association is not desired. The SystemImageGUID may be the NodeGUID of one of the associated nodes if that node is not field-replaceable.
NodeGUID[a]	RO	64	96	GUID of the HCA, TCA, switch, or router itself. All ports on the same node **shall** report the same Node-GUID. Provides a means to uniquely identify a node within a subnet and determine co-location of ports.
PortGUID[b]	RO	64	160	GUID of this port itself. One port within a node can return the NodeGUID as its PortGUID if the port is an integral part of the node and is not field-replaceable.

Table 141 NodeInfo (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
PartitionCap[a]	RO	16	224	Number of entries in the Partition Table for CA, router, and the switch management port. This is at a minimum set to 1 for all nodes including switches.
DeviceID[a]	RO	16	240	Device ID information as assigned by device manufacturer.
Revision[a]	RO	32	256	Device revision, assigned by manufacturer.
LocalPortNum	RO	8	288	The number of the link port which received this SMP.
VendorID[a]	RO	24	296	Device vendor, per IEEE.

a. Value **shall** be the same for all ports on a node.
b. Value **shall** differ for each end port on a CA or router, but the same for all ports of a switch.

14.2.5.4 SWITCHINFO

The SwitchInfo Attribute provides management information specific to switch nodes. It **shall** be implemented by all switches.

Table 142 SwitchInfo

Component	Access	Length (bits)	Offset (bits)	Description
LinearFDBCap	RO	16	0	Number of entries supported in the Linear Unicast Forwarding Table (starting at LID=0x0000 going up). LinearFDBCap = 0 indicates that there is no Linear Forwarding Table.
RandomFDBCap	RO	16	16	Number of entries supported in the Random Unicast Forwarding Table. RandomFDBCap = 0 indicates that there is no Random Forwarding Table.
MulticastFDBCap	RO	16	32	Number of entries supported in the Multicast Forwarding Table (starting at LID=0xC000 going up).
LinearFDBTop	RW	16	48	Indicates the top of the linear forwarding table. Packets received with unicast DLIDs greater than this value are discarded by the switch. A valid LinearFdbTop is less than LinearFdbCap. This component applies only to switches that implement linear forwarding tables and is ignored by switches that implement random forwarding tables.
DefaultPort	RW	8	64	Forward to this port all the unicast packets from the other ports whose DLID does not exist in the random forwarding table, see Chapter 18:: Switches. If set to a non-existent port, subsequent responses may contain any non-existent port number.

Table 142 SwitchInfo (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
DefaultMulticastPrimaryPort	RW	8	72	Forward to this port all the multicast packets from the other ports whose DLID does not exist in the forwarding table, see 18.2.4.3.3 Required Multicast Relay on page 1053. If set to a non-existent port, subsequent responses may contain any non-existent port number.
DefaultMulticastNotPrimaryPort	RW	8	80	Forward to this port all the multicast packets from the Default Primary port whose DLID does not exist in the forwarding table, see 18.2.4.3.3 Required Multicast Relay on page 1053. If set to a non-existent port, subsequent responses may contain any non-existent port number.
LifeTimeValue	RW	5	88	Sets the time a packet can live in the switch, see 18.2.5.4 Transmitter Queueing on page 1057.
PortStateChange	RW	1	93	It is set to one anytime the PortState component in the PortInfo of any ports transitions from Down to Initialize, Initialize to Down, Armed to Down, or Active to Down as a result of link state machine logic. Changes in PortState resulting from SubnSet() do not change this bit. This bit is cleared by writing one, writing zero is ignored.
Reserved	RO	2	94	Reserved
LIDsPerPort	RO	16	96	Specifies the number of LID/LMC combinations that may be assigned to a given external port for switches that support the Random Forwarding table.
PartitionEnforcementCap	RO	16	112	Specifies the number of entries in the partition enforcement table per physical port. Zero indicates that partition enforcement is not supported by the switch.
InboundEnforcementCap	RO	1	128	Indicates switch is capable of partition enforcement on received packets
OutboundEnforcementCap	RO	1	129	Indicates switch is capable of partition enforcement on transmitted packets
FilterRawInboundCap	RO	1	130	Indicates switch is capable of raw packet enforcement on received packets
FilterRawOutboundCap	RO	1	131	Indicates switch is capable of raw packet enforcement on transmitted packets
EnhancedPort0	RO	1	132	When set to 1, indicates switch port 0 supports enhanced functions (TCA port). When set to 0, indicates switch port 0 is a base switch port 0.
Reserved	RO	3	133	Reserved

14.2.5.5 GUIDINFO

The GUIDInfo Attribute provides the means for setting the assigned local scope EUI-64 identifiers of channel adapter, router, and enhanced switch management ports. These local scope EUI-64 identifiers are concatenated with a subnet prefix to form GIDs that are described in 4.1.1 GID Usage and Properties on page 143.

The AttributeModifier is a pointer to a block of 8 GUIDs to which this attribute applies. Valid values are from 0 to 31 and are further limited by the size of the GUIDCap of the port; see 14.2.5 Attributes on page 809. The GUID Block Element at offset zero of the first GUIDBlock (AttributeModifier = 0) is read-only and is a copy of the PortGUID component.

The attribute selected corresponds to the port that received the SMP.

Table 143 GUIDInfo

Component	Access	Length(bits)	Description
GUIDBlock	RW	512	List of 8 GUID Block Elements.

Table 144 GUID Block Element

Component	Length(bits)	Description
GUID	64	GUID to be assigned to port.

14.2.5.6 PORTINFO

The PortInfo Attribute provides port-specific management information. It **shall** be implemented for every port on a node. Note that the values of some PortInfo components vary by node type and by port within a node.

The AttributeModifier selects the port upon which the operation specified by the SMP is performed. For switches, channel adapters, and routers, the range of values between 0 to N, where N is the number of ports and:

- For channel adapters and routers the value of zero indicates that the operation is performed on the port that received the SMP. Otherwise, if the value is non-zero and does not match the port number where the SMP is received, the PortInfo attribute is RO and the M_Key is checked for both the port where the SMP was received and the port selected by the AttributeModifier.
- For switches, a value of zero selects the management port. Otherwise, if the value is non-zero, a physical port is selected.

C14-24.2.1: If PortInfo:Portstate=Down, then

- a SubnGet(PortInfo) **shall** produce valid data for PortInfo:PortState and PortInfo:PortPhysicalState; whether any other component has valid data is vendor-dependent.
- a SubnSet(PortInfo) **shall** make any changes it specifies to PortInfo:PortPhysicalState; any other result is vendor-dependent.

In Table 145: PortInfo, the "Used By" columns indicate how and whether a component is used by a particular type of port: Channel Adapter (CA), Router (Router), Switch external port (Sw Ext.), Base Switch Port 0 (Base SP0) and Enhanced Switch Port 0 (Enh. SP0). The notation used in the columns is: X = required; 0 = must be 0; 1 = must be 1; and blank = unused.

Table 145 PortInfo

Component	Used By					Access	Length (bits)	Offset (bits)	Description
	CA	Router	Sw Ext.	Base SP0	Enh. SP0				
M_Key	X	X		X	X	RW	64	0	The 8-byte management key. See 14.2.4 Management Key on page 806.
GidPrefix	X	X		X	X	RW	64	64	GID prefix for this port.
LID	X	X		X	X	RW	16	128	The base LID of this port.
MasterSMLID	X	X		X	X	RW	16	144	The LID of the master SM that is managing this port.

Table 145 PortInfo (Continued)

Component	Used By					Access	Length (bits)	Offset (bits)	Description
	CA	Router	Sw Ext.	Base SP0	Enh. SP0				
CapabilityMask	X	X		X	X	RO	32	160	Supported capabilities of this node. A bit set to 1 for affirmation of supported capability. 0: Reserved 1: IsSM 2: IsNoticeSupported 3: IsTrapSupported 4: IsOptionalIPDSupported[a] 5: IsAutomaticMigrationSupported 6: IsSLMappingSupported 7: IsMKeyNVRAM (supports M_Key in NVRAM) 8: IsPKeyNVRAM (supports P_Key in NVRAM) 9: IsLEDInfoSupported 10: IsSMdisabled 11: IsSystemImageGUIDSupported 12: IsPKeySwitchExternalPortTrapSupported 13-15: Reserved 16: IsCommunicationManagementSupported 17: IsSNMPTunnelingSupported 18: IsReinitSupported 19: IsDeviceManagementSupported 20: IsVendorClassSupported 21: IsDRNoticeSupported 22: IsCapabilityMaskNoticeSupported 23: IsBootManagementSupported 24: IsLinkRoundTripLatencySupported 25: IsClientReregistrationSupported 26-31: Reserved
DiagCode	X	X		X	X	RO	16	192	Diagnostic code, as described in 14.2.5.6.1 Interpretation of DiagCode on page 832.
M_KeyLeasePeriod	X	X		X	X	RW	16	208	Specifies the initial value of the lease period timer in seconds. The lease period is the length of time that the M_Key Protection bits are to remain non zero after a SubnSet(PortInfo) fails a M_Key check. See 14.2.4 Management Key on page 806.
LocalPortNum	X	X	X	X	X	RO	8	224	The number of the link port which received this SMP.

Table 145 PortInfo (Continued)

Component	Used By					Access	Length (bits)	Offset (bits)	Description
	CA	Router	Sw Ext.	Base SP0	Enh. SP0				
LinkWidthEnabled	X	X	X		X	RW	8	232	Enabled link width, indicated as follows: 0: No State Change; valid only on Set() 1: 1x 2: 4x 3: 1x or 4x 4: 8x 5: 1x or 8x 6: 4x or 8x 7: 1x or 4x or 8x 8: 12x 9: 1x or 12x 10: 4x or 12x 11: 1x or 4x or 12x 12: 8x or 12x 13: 1x or 8x or 12x 14: 4x or 8x or 12x 15: 1x or 4x or 8x or 12x 16 - 254: Reserved 255: Set to LinkWidthSupported value. Changes to this component do not take effect immediately. See InfiniBand Architecture Specification Volume 2, Link/Phy Interface chapter.
LinkWidthSupported	X	X	X		X	RO	8	240	Supported link width, indicated as follows: 1: 1x 3: 1x or 4x 7: 1x or 4x or 8x 11: 1x or 4x or 12x 15: 1x or 4x or 8x or 12x 0, 2, 4-6, 7-10, 12-14, 16-255: Reserved
LinkWidthActive	X	X	X		X	RO	8	248	Currently active link width, indicated as follows: 1: 1x 2: 4x 4: 8x 8: 12x 0, 3, 5-7, 9-255: Reserved

Table 145 PortInfo (Continued)

Component	Used By					Access	Length (bits)	Offset (bits)	Description
	CA	Router	Sw Ext.	Base SP0	Enh. SP0				
LinkSpeedSupported	X	X	X		X	RO	4	256	Supported link speed, indicated as follows: 1: 2.5 Gbps 3: 2.5 or 5.0 Gbps 5: 2.5 or 10.0 Gbps 7: 2.5 or 5.0 or 10.0 Gbps 0, 2, 4, 6, 8-15: Reserved
PortState	X	X	X		X	RW	4	260	Port State. Enumerated as: 0: No State Change; valid only on Set(). 1: Down (includes failed links) 2: Initialize 3: Armed 4: Active 5 - 15: Reserved When writing this field, only legal transitions are valid. See 7.2 Link States on page 168.
PortPhysicalState	X	X	X		X	RW	4	264	0: No state change; valid only on Set() 1: Sleep 2: Polling 3: Disabled 4: PortConfigurationTraining 5: LinkUp 6: LinkErrorRecovery 7: Phy Test 8 - 15: Reserved When writing this field, only values 0, 1, 2, and 3 are valid. Other values are ignored. See InfiniBand Architecture Specification Volume 2, Link/Phy Interface chapter.
LinkDownDefaultState	X	X	X		X	RW	4	268	0: No state change; valid only on Set() 1: Sleep 2: Polling 3 - 15: Reserved See InfiniBand Architecture Specification Volume 2, Link/Phy Interface chapter.
M_KeyProtectBits	X	X		X	X	RW	2	272	See 14.2.4 Management Key on page 806.
Reserved	X	X	X	X	X	RO	3	274	Reserved
LMC	X	X		0	X	RW	3	277	LID mask count for multipath support; its usage is described in 7.11 Subnet Multipathing on page 219.

Table 145 PortInfo (Continued)

Component	Used By					Access	Length (bits)	Offset (bits)	Description
	CA	Router	Sw Ext.	Base SP0	Enh. SP0				
LinkSpeedActive	X	X	X		X	RO	4	280	Currently active link speed, indicated as follows: 1: 2.5 Gbps 2: 5.0 Gbps 4: 10.0 Gbps 0, 3, 5-15: reserved
LinkSpeedEnabled	X	X	X		X	RW	4	284	Enabled link speed, indicated as follows: 0: No State Change; valid only on Set() 1: 2.5 Gbps 2: 5.0 Gbps 3: 2.5 or 5.0 Gbps 4: 10.0 Gbps 5: 2.5 or 10.0 Gbps 6: 5.0 or 10.0 Gbps 7: 2.5 or 5.0 or 10.0 Gbps 8-14 Reserved 15: Set to LinkSpeedSupported values Changes to this component do not take effect immediately. See InfiniBand Architecture Specification Volume 2, Link/Phy Interface chapter.
NeighborMTU	X	X	X		X	RW	4	288	Active maximum MTU enabled on this port for transmit: 1: 256 2: 512 3: 1024 4: 2048 5: 4096 0, 6 - 15: reserved
MasterSMSL	X	X		X	X	RW	4	292	The administrative SL of the master SM that is managing this port.
VLCap	X	X	X		X	RO	4	296	Virtual Lanes supported on this port, indicated as follows: 1: VL0 2: VL0, VL1 3: VL0 - VL3 4: VL0 - VL7 5: VL0 - VL14 0, 6 - 15: reserved

Table 145 PortInfo (Continued)

Component	Used By					Access	Length (bits)	Offset (bits)	Description
	CA	Router	Sw Ext.	Base SP0	Enh. SP0				
InitType	X	X		X	X	RO	4	300	Optional; **shall** be 0 if not implemented. Type of initialization requested by this port before SM moves it to Active or Armed state. See 14.4.4 Node Reinitialization on page 871. • bit 0: NoLoad. 0 = Port is requesting that its attributes be initialized (see 14.4.3 Initialization Actions on page 868). 1 = Port is requesting that no data be loaded into its attributes at all, asserting that the last-loaded data still exists and is valid. • bit 1: PreserveContent. 0 = Port makes no request regarding content of the data that is loaded into its attributes. 1 = Port is requesting that all such data, if loaded, be set to the most recent content loaded by the SM. • bit 2: PreservePresence. 0 = Port is requesting that all settable SA attributes referencing this port (see Table 190 Subnet Administration Attribute / Method Map on page 890) be removed prior to activating this port and Report()s of in/out of service (trap numbers 64/65) be sent. 1 = Port is requesting that all such data be preserved, and Report()s of in/out of service (trap numbers 64/65) not be sent for this port. • bit 3: DoNotResuscitate. 0 = bits 0, 1, and 2 of this field are valid; initialization of this port should begin based on their values. 1 = bits 0, 1, and 2 are not valid; port is requesting that reinitialization of this port, and any Report()s of in/out of service (trap numbers 64/65) be delayed until this bit is set to 0.
VLHighLimit	X	X	X		X	RW	8	304	Limit of High Priority component of VL Arbitration Table, as defined in 7.6.9 VL Arbitration and Prioritization on page 188.
VLArbitrationHighCap	X	X	X		X	RO	8	312	VL/Weight pairs supported on this port in the VLArbitration table for high priority. **Shall** be 1 to 64 if more than one data VL is supported on this port, 0 otherwise. See 7.6.9 VL Arbitration and Prioritization on page 188.

Table 145 PortInfo (Continued)

Component	Used By					Access	Length (bits)	Offset (bits)	Description
	CA	Router	Sw Ext.	Base SP0	Enh. SP0				
VLArbitrationLowCap	X	X	X		X	RO	8	320	VL/Weight pairs supported on this port in the VLArbitration table for low priority. **Shall** be N to 64 if more than one data VL is supported on this port, 0 otherwise, N being the number of data VLs supported. See 7.6.9 VL Arbitration and Prioritization on page 188.
InitTypeReply	X	X		X	X	RW	4	328	Written by the SM prior to changing the port to Active or Armed state. Optional; **shall** be set to all 0s if not implemented (PortInfo:CapabilityMask.IsReinitSupported=0 or SA's ClassPortInfo:CapabilityMask.IsReinitSupported=0). Indicates the type of initialization performed. See 14.4.4 Node Reinitialization on page 871. • bit 0: NoLoadReply. 0 = Port attributes were initialized (see 14.4.3 Initialization Actions on page 868). 1 = No data was loaded into the port attributes. • bit 1: PreserveContentReply. 0 = No information is available regarding content of the data loaded into the port attributes. 1 = The data loaded into the port attributes was set to the content most recently loaded by the SM. • bit 2: PreservePresenceReply. 0 = All settable SA attributes referencing this port (see Table 190 Subnet Administration Attribute / Method Map on page 890) were removed prior to activating this port and Report()s of in/out of service (trap numbers 64/65) were sent. 1 = All such data existing when this port was last active was not removed and Report()s of in/out of service (trap numbers 64/65) were not sent. • bit 3: Reserved.
MTUCap	X	X	X		X	RO	4	332	Maximum MTU supported by this port. 1: 256 2: 512 3: 1024 4: 2048 5: 4096 0, 6 - 15: reserved

Table 145 PortInfo (Continued)

Component	Used By					Access	Length (bits)	Offset (bits)	Description
	CA	Router	Sw Ext.	Base SP0	Enh. SP0				
VLStallCount			X			RW	3	336	Specifies the number of sequential packets dropped that causes the port to enter the VLStalled state. The result of setting the VLStallCount to a value of zero is undefined. Refer to 18.2.5.4 Transmitter Queueing on page 1057 for details.
HOQLife		X	X			RW	5	339	Sets the time a packet can live at the head of a VL queue. Refer to 18.2.5.4 Transmitter Queueing on page 1057 for details.
OperationalVLs	X	X	X		X	RW	4	344	Virtual Lanes operational on this port, indicated as follows: 0: No change; valid only on Set() 1: VL0 2: VL0, VL1 3: VL0 - VL3 4: VL0 - VL7 5: VL0 - VL14 6 - 15: reserved Changing OperationalVLs in certain PortStates may cause flow control update errors which may initiate Link/Phy retraining.
PartitionEnforcementInbound			X			RW	1	348	Indicates support of optional partition enforcement. If set to one, enables partition enforcement on packets received on this port. Zero disables partition enforcement on packets received on this port.
PartitionEnforcementOutbound			X			RW	1	349	Indicates support of optional partition enforcement. If set to one, enables partition enforcement on packets transmitted from this port. Zero disables partition enforcement on packets transmitted on this port.
FilterRawInbound			X			RW	1	350	Indicates support of optional raw packet enforcement. If set to 1, raw packets arriving on this port are discarded. If set to 0, inbound raw packets are not discarded; they are processed normally. See C18-16: on page 1046.
FilterRawOutbound			X			RW	1	351	Indicates support of optional raw packet enforcement. If set to 1, raw packets departing on this port are discarded. If set to 0, outbound raw packets are not discarded; they are processed normally. See C18-52: on page 1056.

Table 145 PortInfo (Continued)

Component	Used By					Access	Length (bits)	Offset (bits)	Description
	CA	Router	Sw Ext.	Base SP0	Enh. SP0				
M_KeyViolations	X	X		X	X	RW	16	352	Counts the number of SMP packets that have been received at this port that have had invalid M_Keys, since power-on or reset. Increments till count reaches all 1s and then **shall** be set back to zero to re-enable incrementing. Setting this component to any value other than zero results in undefined behavior; however, it is recommended that any attempt to set the counter to a non-zero value results in it being left unchanged.
P_KeyViolations	X	X		X	X	RW	16	368	Counts the number of packets that have been received at this port that have had invalid P_Keys, since power-on or reset. Refer to 10.9.4 Bad P_Key Trap and P_Key Violations Counter (Optional) on page 526 for usage description. Increments till count reaches all 1s and then **shall** be set back to zero to re-enable incrementing. Setting this component to any value other than zero results in undefined behavior; however, it is recommended that any attempt to set the counter to a non-zero value results in it being left unchanged.
Q_KeyViolations	X	X		X	X	RW	16	384	Counts the number of packets that have been received at this port that have had invalid Q_Keys, since power-on or reset. See 10.2.5 Q_Keys on page 439 for usage description. Increments till count reaches all 1s and then **shall** be set back to zero to re-enable incrementing. Setting this component to any value other than zero results in undefined behavior; however, it is recommended that any attempt to set the counter to a non-zero value results in it being left unchanged.
GUIDCap	X	X		X	X	RO	8	400	Number of GUID entries supported in the GUIDInfo attribute for this port.
ClientReregister	X	X		X	X	RW	1	408	Optional: **shall** be 0 if not implemented (PortInfo:CapabilityMask.IsClientReregistrationSupported = 0). Used by SM to request endnode client reregistration of SA subscriptions. See 14.4.11 Client Reregistration on page 881.
Reserved	X	X	X	X	X	RO	2	409	Reserved

Table 145 PortInfo (Continued)

Component	Used By					Access	Length (bits)	Offset (bits)	Description
	CA	Router	Sw Ext.	Base SP0	Enh. SP0				
SubnetTimeOut	X	X		X	X	RW	5	411	Specifies the maximum expected subnet propagation delay, which depends upon the configuration of the switches, to reach any other port in the subnet and **shall** also be used to determine the maximum rate which SubnTrap()s can be sent from this port. The duration of time is calculated based on (4.096 µsec * $2^{SubnetTimeOut}$).
Reserved	X	X	X	X	X	RO	3	416	Reserved
RespTimeValue	X	X		X	X	RO	5	419	Specifies the expected maximum time between the port reception of a SMP and the transmission of the associated response. The duration of time is calculated based on (4.096 µsec * $2^{RespTimeValue}$).
LocalPhyErrors	X	X	X		X	RW	4	424	Threshold value. When the count of marginal link errors exceeds this threshold, the local link integrity error **shall** be detected as described in 7.12.2 Error Recovery Procedures on page 221.
OverrunErrors	X	X	X		X	RW	4	428	Threshold value. When the count of buffer overruns over consecutive flow control update periods exceeds this threshold as described in 7.12 Error detection and handling on page 219, the excessive buffer overrun error **shall** be detected as described in that section.
MaxCreditHint	X	X	X		X	RO	16	432	Optional; shall be 0 if not implemented (PortInfo:CapabilityMask.IsLinkRoundTripLatencySupported = 0). This value provides a vendor-dependent indication of the maximum number of credits available per VL on the Port.
Reserved	X	X	X	X	X	RO	8	448	Reserved

Table 145 PortInfo (Continued)

Component	Used By: CA	Router	Sw Ext.	Base SP0	Enh. SP0	Access	Length (bits)	Offset (bits)	Description
LinkRoundTripLatency	X	X	X		0	RO	24	456	Optional; shall be 0 if not implemented (PortInfo:CapabilityMask.IsLinkRoundTripLatencySupported = 0). This value represents a measurement of the round-trip latency of the link attached to this port. It is an unsigned 24-bit integer counting 4 nsec. intervals. This value might not be accurate to better than +/- 4 nsec. A value of 0 is valid when implemented, and indicates a latency of up to 4 nsec. LinkRoundTripLatency is reset to 0xFFFFFF whenever this port transitions to PortState = Down. Each time a link heartbeat reply (ACK) is received on this port when PortState is not Down, this value is set to the minimum of (a) the prior value of this attribute; and (b) the time elapsed since the corresponding heartbeat (SND) was sent. Note that while the value of this component will never increase, it may change over time due to varying queueing delays associated with traffic load. Note that this component is always 0 for Enhanced SP 0. See InfiniBand Architecture Volume 2, Link/Phy Interface chapter.

a. IsOptionallPDSupported is ignored, and support for the optional IPD values (see Table 315, "Static Rate Control IPD Values," on page 1029) is assumed to be present, if PortInfo:LinkSpeedSupported is other than 1, or if PortInfo:LinkWidthSupported is other than 1, 3, or 11.

14.2.5.6.1 INTERPRETATION OF DIAGCODE

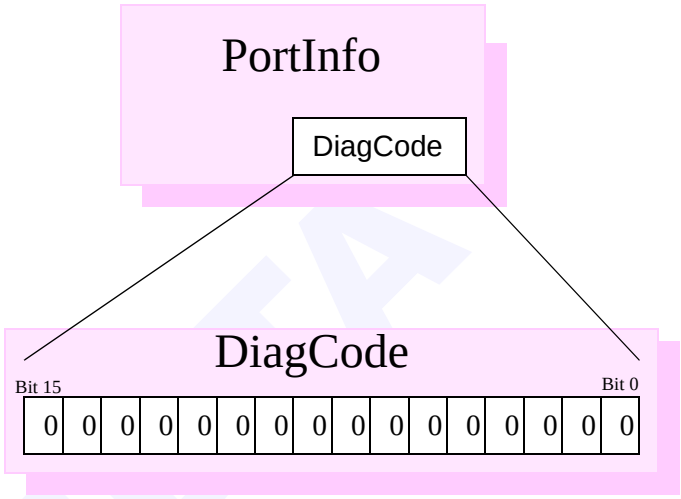
Endports of an IBA network will have functions attached to the non-IBA side of those endports. The 16-bit PortInfo:DiagCode field provides both generic and vendor-specific diagnostic status for those non-IBA functions. It is valid only on endports and all bits set to zero means the function status is good. Any non-zero value means there are possible error conditions. Additional status information specific to a particular management class may also be provided, depending on the function, but PortInfo:DiagCode is always present.

The PortInfo:DiagCode can provide three levels of diagnostic data:

- A high level, universal set of status codes. A PortInfo:DiagCode of all zeroes indicates no exception conditions exist. Nonzero values of bits 3-0 of PortInfo:DiagCode have the same meanings for all non-IBA attached functions; these meanings are documented below in Table 146 Standard Encoding of DiagCode Bits 3-0 on page 833.

- An optional, high level vendor-specific diagnostic code in bits 14-4 of PortInfo:DiagCode. Interpretation of this field requires knowledge of the diagnostic codes supported by the functions behind the endport.
- An optional, more detailed vendor-specific port attribute pointed to by PortInfo:DiagCode. Availability of this information is indicated by bit 15 of PortInfo:DiagCode and the pertinent port attribute is then pointed to by bits 14-4.

Figure 190 DiagCode Fields on page 833 summarizes the structure and interpretation of PortInfo:DiagCode fields.



Bits 3-0 are reserved for the universal diagnostic codes.

Bits 14-4 provide vendor-specific diagnostic codes.

Bit 15 provides a means to chain vendor diagnostic attributes together.

Figure 190 DiagCode Fields

The error information in bits 3-0 is interpreted in a standard fashion for all endports as shown in Table 146 Standard Encoding of DiagCode Bits 3-0 on page 833.

Table 146 Standard Encoding of DiagCode Bits 3-0

DiagCode Bits 3-0	Description
0x0	Function Ready
0x1	Performing Self Test
0x2	initializing
0x3	soft error - function has a non-fatal error and may be used
0x4	hard error - function may not be used
0x5 - 0xF	reserved

Bits 14-4 of PortInfo:DiagCode are used for vendor-specific modifiers to the standard diagnostic information, as shown in Figure 191 Example of DiagCode Bits on page 834.

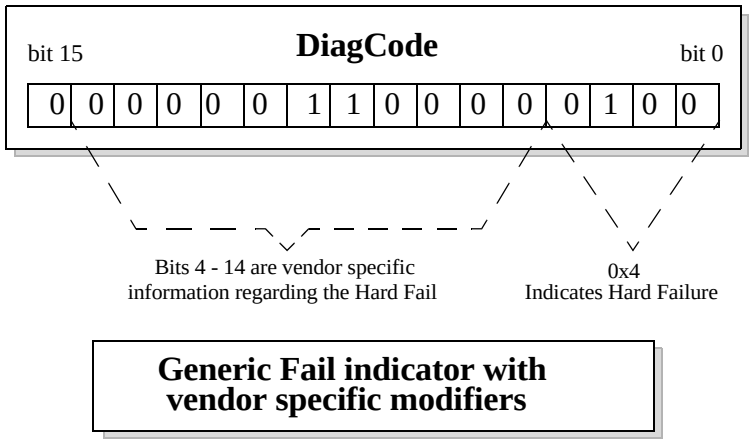


Figure 191 Example of DiagCode Bits

If bit 15, the IndexForward bit, is zero (0), bits 14-4 of the PortInfo:DiagCode represent a vendor-specific diagnostic code. Interpretation of the returned information is outside the scope of this specification. Further diagnostic information might be provided in known locations in one or more vendor-specific attributes.

If the IndexForward bit is set, bits 14-4 of the PortInfo:DiagCode field are used to index into the VendorDiag Attribute data for vendor-specific diagnostic information. This allows dynamic chaining of diagnostic information based on the type of exception. Bits 14-4 are interpreted as an AttributeModifier to be specified with an SubnGet(VendorDiag) to the port being examined as defined in 14.2.5.14 VendorDiag on page 840.

14.2.5.7 P_KEYTABLE

The P_KeyTable Attribute provides the means for assigning the P_Keys for ports.

The AttributeModifier is divided in two halves:

- The least significant 16 bits are a pointer to a block of 32 P_Key entries to which this Attribute applies. Valid values are 0 - 2047, and are further limited by the size of the P_Key table for that node (specified by the PartitionCap for CAs, routers, and switch management ports or PartitionEnforcementCap for external ports on switches). See 14.2.5 Attributes on page 809.
- For switches, the upper 16 bits select the switch port, where valid values are 1 - 254 to select physical ports and zero to select the switch management port.

For CA and router, the upper 16 bits are ignored and the operation is performed on the port that received the SMP.

Table 147 P_KeyTable

Component	Access	Length (bits)	Description
P_KeyTable Block	RW	512	List of 32 P_Key Block Elements.

Table 148 P_Key Block Element

Component	Length (bits)	Description
Membership-Type	1	If set to zero, the P_Key is limited type and the endnode may accept a packet with a matching full P_Key, but may not accept a packets with a matching limited P_Key. If set to one, the P_Key is full type and the endnode may receive packets with matching full or limited P_Key. A full description is in 10.9.1.1 Limited and Full Membership on page 524.
P_KeyBase	15	Base value of the P_Key that the endnode will use to check against incoming packets.

14.2.5.8 SLtoVLMappingTable

The SLtoVLMappingTable Attribute provides the means for setting the SL to VL Mapping of a switch, CA, and router and its usage is described in 7.6.6 VL Mapping Within a Subnet on page 186.

For a switch, this attribute is specific to an input port / output port combination to which the specific SL to VL mapping applies:

- bits 31-16 must be zero.
- bits 15-8 of the AttributeModifier specify the input port which can be 1 to N, where N selects the physical port or 0 to indicate that the input port is the management port.
- bits 7-0 of the AttributeModifier specify the output port which can be 1 to N, where N selects the physical port. For implementations supporting enhanced switch port 0, N = 0 indicates the management port.

For CA and router, this attribute corresponds to the port receiving the SMP (the AttributeModifier is ignored)..

Table 149 SLtoVLMappingTable

Component	Access	Length(bits)	Offset (bits)	Description
SL0toVL	RW	4	0	The number of the VL on which packets using SL0 are output. 15 forces the packets to be dropped.

Table 149 SLtoVLMappingTable (Continued)

Component	Access	Length(bits)	Offset (bits)	Description
SL1toVL	RW	4	4	The VL associated with SL1
SL2toVL	RW	4	8	The VL associated with SL2
SL3toVL	RW	4	12	The VL associated with SL3
SL4toVL	RW	4	16	The VL associated with SL4
SL5toVL	RW	4	20	The VL associated with SL5
SL6toVL	RW	4	24	The VL associated with SL6
SL7toVL	RW	4	28	The VL associated with SL7
SL8toVL	RW	4	32	The VL associated with SL8
SL9toVL	RW	4	36	The VL associated with SL9
SL10toVL	RW	4	40	The VL associated with SL10
SL11toVL	RW	4	44	The VL associated with SL11
SL12toVL	RW	4	48	The VL associated with SL12
SL13toVL	RW	4	52	The VL associated with SL13
SL14toVL	RW	4	56	The VL associated with SL14
SL15toVL	RW	4	60	The VL associated with SL15

14.2.5.9 VLARBITRATIONTABLE

The VLArbitrationTable Attribute provides the means for setting the VL Arbitration for ports on CAs, routers and switches; its usage is described in 7.6.9 VL Arbitration and Prioritization on page 188.

The AttributeModifier is divided in two halves. The upper 16 bits specify the part of the tables that is accessed.

- 1 - lower 32 entries of the low priority VL Arbitration Table.
- 2 - upper 32 entries of the low priority VL Arbitration Table.
- 3 - lower 32 entries of the high priority VL Arbitration Table.
- 4 - upper 32 entries of the high priority VL Arbitration Table.
- 0, 5-65535 - reserved.

For switches, the least significant 16 bits of the AttributeModifier specify the external port or enhanced (not base) port 0 in bits 7-0; and bits 15-8 are reserved.

For CA and router, this attribute corresponds to the port receiving the SMP (the lower 16 bits of the AttributeModifier are ignored).

The size of the VLArbitrationTable, based on PortInfo:VLArbitrationHighCap and PortInfo:VLArbitrationLowCap, further limits the valid values. See 14.2.5 Attributes on page 809

Table 150 VLArbitrationTable

Component	Access	Length (bits)	Offset (bits)	Description
VL/Weight pairs	RW	512	0	Lists of 32 VL/Weight Block elements, for which there may be up to 64 in total for a given priority.

Table 151 VL/Weight Block Element

Component	Length (bits)	Offset (bits)	Description
Reserved	4	0	Reserved
VL	4	4	VL associated with element.
Weight	8	8	Weight associated with element, as defined in 7.6.9 VL Arbitration and Prioritization on page 188, zero indicates that this element is skipped.

14.2.5.10 LINEARFORWARDINGTABLE

The LinearForwardingTable Attribute provides the means for setting the linear forwarding table of a switch for the Unicast LIDs.

The AttributeModifier is a pointer to a block of 64 LIDs to which this attribute applies. Valid values are from 0 to 767, and are further limited by the size of the LinearForwardingTable of the switch. An implementation may choose either SwitchInfo:LinearFDBCap or SwitchInfo:LinearFDBTop to denote the size of the LinearForwardingTable. See 14.2.5 Attributes on page 809.

Table 152 LinearForwardingTable

Component	Access	Length(bits)	Description
LinearForwarding-Table Block	RW	512	List of 64 Port Block Elements.

Table 153 Port Block Element

Component	Length(bits)	Description
Port	8	Port to which packets with the LID corresponding to this entry are to be forwarded.

14.2.5.11 RANDOMFORWARDINGTABLE

The RandomForwardingTable Attribute provides the means for setting the random forwarding table of a switch for the Unicast LIDs.

The AttributeModifier is a pointer to a block of 16 LID/port pairs to which this Attribute applies. Valid values are from 0 to 3071, and are further limited by the size of the RandomForwardingTable of the switch based on SwitchInfo:RandomFDBCap. See 14.2.5 Attributes on page 809. If an invalid port number is written into an entry that has the Valid bit set to 1, packets sent to the LIDs specified in the entry will be discarded and that entry's Port shall be read back as 0xFF to indicate that an invalid port number was used. If two or more entries specify LID ranges that overlap based on LMCs, and the Valid bit for these entries is set to 1, switch forwarding behavior for packets sent to these LIDs is implementation-dependent.

Table 154 RandomForwardingTable

Component	Access	Length(bits)	Description
RandomForwardingTable Block	RW	512	List of 16 LID/Port Block Elements.

Table 155 LID/Port Block Element

Component	Length(bits)	Offset (bits)	Description
LID	16	0	Base LID.
Valid	1	16	This LID/Port pair is valid. Note that setting this parameter to 0 allows the removal of entries.
LMC	3	17	the LMC of this LID.
Reserved	4	20	Reserved
Port	8	24	Port to which packets with this LID/LMC corresponding to this entry are to be forwarded.

14.2.5.12 MULTICASTFORWARDINGTABLE

This MulticastForwardingTable Attribute provides the means for setting the multicast forwarding table of a switch.

The nine low-order bits of the AttributeModifier are a pointer to a block of 32 PortMask entries to which this attribute applies. Valid values are limited by the size of the MulticastForwardingTable of the switch based on SwitchInfo:MulticastFDBCap; see 14.2.5 Attributes on page 809. See Figure 192 MulticastForwardingTable Bit Layout on page 839 for how blocks of PortMasks are mapped into MulticastForwardingTable entries.

The four high-order bits of the AttributeModifier indicate the position (p) of the 16-bit PortMask entry of this Attribute. Each PortMask entry specifies only 16 bits of the 256 possible bits of a port mask of a maximum size switch. (see Figure 192 MulticastForwardingTable Bit Layout on page 839). Valid values of the position bits are limited by the number of ports on the switch. PortMask bits in a block that are beyond the number of switch ports are ignored on write and read back as zero.

The remaining 18 bits (bits 27:9) of the AttributeModifier **shall** be set to zero.

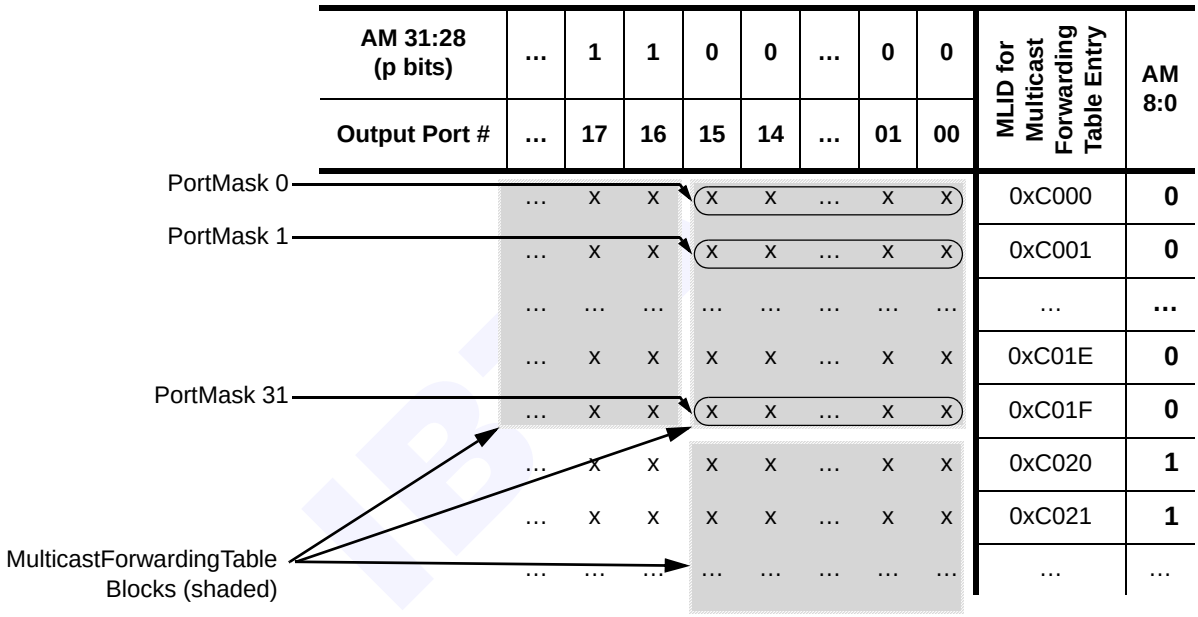


Figure 192 MulticastForwardingTable Bit Layout

.

Table 156 MulticastForwardingTable

Component	Access	Length(bits)	Description
MulticastFor-wardingTable Block	RW	512	List of 32 PortMask Block Elements.

Table 157 PortMask Block Element

Component	Length(bits)	Description
PortMask	16	16 bits starting at position 16×p of the PortMask associated with this LID where each bit in the PortMask **shall** refer to a port number that is equal to 16×p plus the position of the bit within the PortMask. For example, when p equals zero, the least significant bit of the PortMask refers to port 0 and when p equals one, the least significant bit of the PortMask refers to port 16. An incoming packet with this LID is forwarded to all ports for which the bit in the PortMask is set to 1.

14.2.5.13 SMINFO

The SMInfo attribute is used by Subnet Managers to exchange information during subnet discovery and polling as described in 14.4 Subnet Manager on page 859. This attribute **shall** be available on a port where a Subnet Manager resides.

Table 158 SMInfo

Component	Access	Length (bits)	Offset10 (bits)	Description
GUID	RO	64	0	PortGUID of the port where the SM resides.
SM_Key	RO	64	64	Key of this SM. This is shown as 0 unless the requesting SM is authenticated (see 14.4.7 Authentication on page 878).
ActCount	RO	32	128	Counter that increments each time the SM issues an SMP or performs other management activities. Used as a “heartbeat” indicator by standby SMs.
Priority	RO	4	160	Administratively assigned priority for this SM. Can be reset by master SM. Zero is lowest priority. An out-of-band mechanism for setting this value **shall** be provided. The default value, if not set by the out-of-band mechanism, **shall** be zero.
SMState	RO	4	164	Enumerated value indicating this SM’s state. Enumerated as follows: 0 - not active 1 - discovering 2 - standby 3 - master 4-15 - Reserved

14.2.5.14 VENDORDIAG

The VendorDiag Attribute provides a way to obtain vendor specific diagnostic information. The interpretation of the VendorDiag:DiagData is spe-

cific to the port in question. It is accessible from ports on CAs, routers, and the management port on a switch.

Table 159 VendorDiag

Component	Access	Length (bits)	Offset (bits)	Description
NextIndex	RO	16	0	Next AttributeModifier to get to diagnostic Info. Set to zero if this is last or only diagnostic data.
DiagData	RO	496	16	Vendor specific diagnostic information. Format is undefined.

14.2.5.6.1 Interpretation of DiagCode on page 832 describes how the PortInfo:DiagCode forwarding mechanism is used to obtain the address modifier for the VendorDiag attribute during interpretation of diagnostic codes. An example of the use of the IndexForward bit, bit 15 of the PortInfo:DiagCode component, is shown in Figure 193 Index Forwarded Diagnostic Information on page 841.

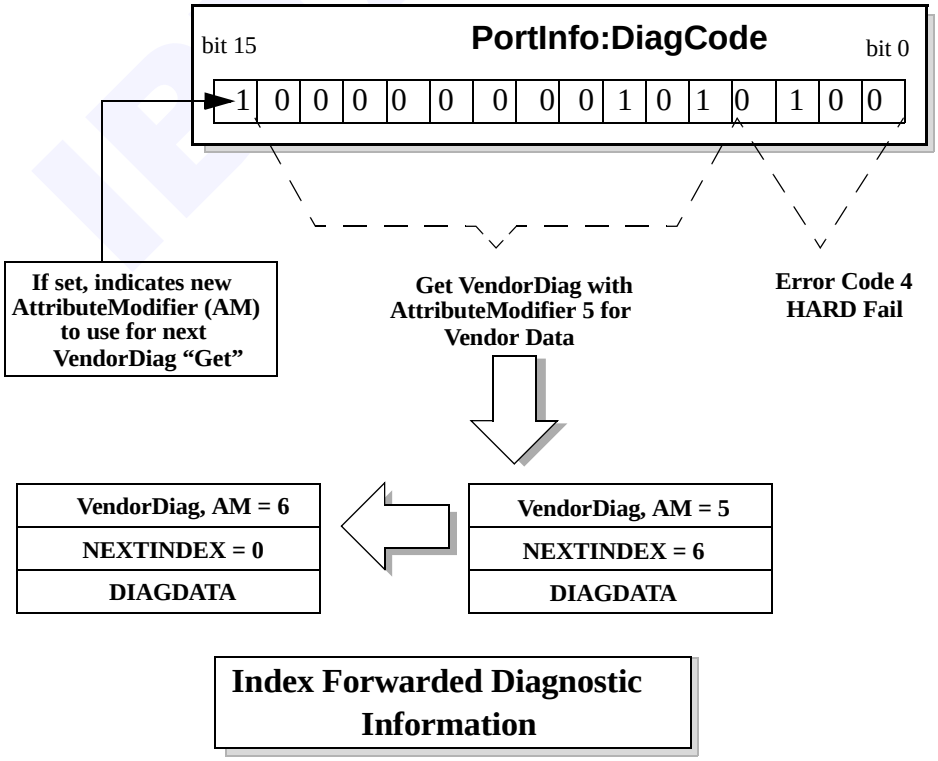


Figure 193 Index Forwarded Diagnostic Information

In the above example the PortInfo:DiagCode with the IndexForward bit set indicates that VendorDiag AttributeModifier 5 of this port contains vendor-specific diagnostic information. When VendorDiag AttributeModifier 5 is retrieved, the VendorDiag:NextIndex value indicates more data at AttributeModifier 6. The retrieval of AttributeModifier 6 returns a VendorDiag:NextIndex of 0, indicating the end of the diagnostic data.

14.2.5.15 LEDINFO

The LedInfo Attribute provides the ability to turn on or off a LED optionally provided by a CA, router, and switch using SMPs. This LED is not specified and the implementation of this LED is vendor-specific. It has no association with LEDs that are specified by this or other volumes of the IB specification. A CA, router, and switch **shall** indicate its support of this attribute in the PortInfo:CapabilityMask.

Table 160 LedInfo

Component	Type	Length (bits)	Offset (bits)	Description
LedMask	RW	1	0	Set to 1 for LED on, and 0 for LED off. The response packet **shall** indicate actual LED state.
Reserved	RO	31	1	Reserved

14.2.6 SUBNET MANAGEMENT MAD STATUS

This section provides a consolidated interpretation of a large number of status-code-related compliance statements in the IBTA Specification Volume 1, Chapters 13, 14, and 16. It is provided to aid interoperability. Note, however, that this document is not compliance.

Following the error condition definitions and their respective error status handling specified here satisfies the minimum requirement for being compliant to all the IB compliance rules relating to the MAD Status. Any additional error conditions and error handling that are beyond what are specified here, without violating any IB compliance rules, can be considered optional to the implementation.

14.2.6.1 STATUS PRECEDENCE

In this implementation approach, there is an implicit precedence among various status settings. The higher the status value, the lower its precedence. This status precedence implementation is only one of many possible implementations. It does not violate nor override any of the compliance rules defined. Other implementations may return an error status without following this status precedence. Instead, if multiple errors

were detected, any one of the error status may be returned in the GetResp().

Table 161 Status Precedence

Precedence (0x1 is highest)	Status[4:2]	Violations
1	0x1	Bad version.
2	0x2	The method specified is not supported.
3	0x3	The method/attribute combination is not supported.
4	0x7	One or more fields in the attribute or attribute modifier contain an invalid value.
5	0x0	None of above violations were found.

14.2.6.2 SMP VERSION NOT SUPPORTED (STATUS_FIELD[4:2] = 0X1)

Table 162 Version Errors in SMPs

Fields	Invalid Values
BaseVersion	outside the vendor specification
ClassVersion	outside the vendor specification
BaseVersion + ClassVersion	outside the vendor specification

14.2.6.3 SMP METHOD NOT SUPPORTED (STATUS_FIELD[4:2] = 0X2)

Upon receiving a request packet with a method that is unsupported by the receiving entity, any of the following are acceptable:

- Packet is silently discarded without a response
- Packet is returned as status 2 with the R bit in the method field set to one without regarding the overall meaning of the method field.
- Packet is returned as status 2 with the GetResp() method

14.2.6.4 SMP METHOD/ATTRIBUTE COMBINATION NOT SUPPORTED (STATUS_FIELD[4:2] = 0X3)

Table 163 Subnet Management Attribute / Method Map Errors

Attribute	Get	Set	Trap	TrapRepress
Notice	error if PortInfo:CapabilityMask.IsNoticeSupported = 0, otherwise valid	error if PortInfo:CapabilityMask.IsNoticeSupported = 0, otherwise valid	valid	valid

Table 163 Subnet Management Attribute / Method Map Errors (Continued)

Attribute	Get	Set	Trap	TrapRepress
NodeDescription	valid	error	error	error
NodeInfo	valid	error	error	error
SwitchInfo	valid if NodeInfo:NodeType is Switch, otherwise error	valid if NodeInfo:NodeType is Switch, otherwise error	error	error
GUIDInfo	valid	valid	error	error
PortInfo	valid	valid	error	error
P_KeyTable	valid	valid	error	error
SLtoVLMappingTable	error if PortInfo:CapabilityMask.IsSLMappingSupported = 0, otherwise valid	error if PortInfo:CapabilityMask.IsSLMappingSupported = 0, otherwise valid	error	error
VLArbitrationTable	error if PortInfo:VLCap = 1, otherwise valid	error if PortInfo:VLCap = 1, otherwise valid	error	error
LinearForwardingTable	error if SwitchInfo:LinearFDBCap = 0 or NodeInfo:NodeType is not Switch, otherwise valid	error if SwitchInfo:LinearFDBCap = 0 or NodeInfo:NodeType is not Switch, otherwise valid	error	error
RandomForwardingTable	error if SwitchInfo:RandomFDBCap = 0 or NodeInfo:NodeType is not Switch, otherwise valid	error if SwitchInfo:RandomFDBCap = 0 or NodeInfo:NodeType is not Switch, otherwise valid	error	error
MulticastForwardingTable	error if SwitchInfo:MulticastFDBCap = 0 or NodeInfo:NodeType is not Switch, otherwise valid	error if SwitchInfo:MulticastFDBCap = 0 or NodeInfo:NodeType is not Switch, otherwise valid	error	error
SMInfo	error if PortInfo:CapabilityMask.IsSMdisabled = 0 AND PortInfo:CapabilityMask.IsSM = 0, otherwise valid[a]	error if PortInfo:CapabilityMask.IsSMdisabled = 0 AND PortInfo:CapabilityMask.IsSM = 0, otherwise valid[a]	error	error
VendorDiag	valid	error	error	error
LedInfo	error if PortInfo:CapabilityMask.IsLEDInfoSupported = 0, otherwise valid	error if PortInfo:CapabilityMask.IsLEDInfoSupported = 0, otherwise valid	error	error

Table 163 Subnet Management Attribute / Method Map Errors (Continued)

Attribute	Get	Set	Trap	TrapRepress
Vendor Specific attributes with AttributeID ranging from 0xFF00 to 0xFFFF	error if Vendor Specific attributes are not supported by this method	error if Vendor Specific attributes are not supported by this method	error if Vendor Specific attributes are not supported by this method	error if Vendor Specific attributes are not supported by this method
None of above	error	error	error	error

a. packet discard if PortInfo:CapabilityMask.IsSMdisabled = 1.

14.2.6.5 SMP ATTRIBUTEMODIFIER ERRORS (STATUS_FIELD[4:2] = 0X7)

Table 164 SMP AttributeModifier Errors

Attribute	Invalid AttributeModifier for Switch with BSP0	Invalid AttributeModifier for CA/Router/Switch with ESP0
Notice	AM[31:0] > 0x0	AM[31:0] > 0x0
NodeDescription	AM[31:0] > 0x0	AM[31:0] > 0x0
NodeInfo	AM[31:0] > 0x0	AM[31:0] > 0x0
SwitchInfo[a]	AM[31:0] > 0x0	AM[31:0] > 0x0
GUIDInfo	AM[31:0] > ((PortInfo:GUIDCap-1) / 8)[b]	AM[31:0] > ((PortInfo:GUIDCap-1) / 8)[b.]
PortInfo[c]	AM[31:0] > NodeInfo:NumPorts	AM[31:0] > NodeInfo:NumPorts
P_KeyTable	AM[31:16][a.] > NodeInfo:NumPorts	AM[31:16][a.] > NodeInfo:NumPorts
	AM[15:0] > ((NodeInfo:PartitionCap or SwitchInfo:PartitionEnforcementCap-1) / 32)[b.]	AM[15:0] > ((NodeInfo:PartitionCap or SwitchInfo:PartitionEnforcementCap-1) / 32)[b.]
SLtoVLMapping-Table[a.]	AM[15:8] (input port) > NodeInfo:NumPorts	(AM[15:8] (input port) > NodeInfo:Num-Ports)
	AM[7:0] (output port) > NodeInfo:NumPorts	(AM[7:0] (output port) > NodeInfo:Num-Ports)
	AM[7:0] (output port) = 0x0	

Table 164 SMP AttributeModifier Errors (Continued)

Attribute	Invalid AttributeModifier for Switch with BSP0	Invalid AttributeModifier for CA/Router/Switch with ESP0
VLArbitrationTable	(AM[7:0] (output port) > NodeInfo:Num-Ports)[a.]	(AM[7:0] (output port) > NodeInfo:Num-Ports)[a.]
	(AM[7:0] (output port) = 0x0)[a.]	
	if (PortInfo:VLArbitrationHighCap ≤ 32), AM[31:16]!=0x3	if (PortInfo:VLArbitrationHighCap ≤ 32), AM[31:16]!=0x3
	if (PortInfo:VLArbitrationLowCap ≤ 32), AM[31:16]!=0x1	if (PortInfo:VLArbitrationLowCap ≤ 32), AM[31:16]!=0x1
	if (33 ≤ PortInfo:VLArbitrationHighCap ≤ 64), AM[31:16]!= 0x3,0x4	if (33 ≤ PortInfo:VLArbitrationHighCap ≤ 64), AM[31:16]!= 0x3,0x4
	if (33 ≤ PortInfo:VLArbitrationLowCap ≤ 64), AM[31:16]!= 0x1,0x2	if (33 ≤ PortInfo:VLArbitrationLowCap ≤ 64), AM[31:16]!= 0x1,0x2
LinearForwarding-Table[a.]	AM[31:0] > (either SwitchInfo:LinearFDB-Cap or[d] (SwitchInfo:LinearFDBTop-1)) / 64)[b.]	AM[31:0] > (either SwitchInfo:LinearFDB-Cap or[d.] (SwitchInfo:LinearFDBTop-1)) / 64)[b.]
	AM[31:0] > 767	AM[31:0] > 767
RandomForwarding-Table[a.]	AM[31:0] > ((SwitchInfo:RandomFDBCap-1) / 16)[b.]	AM[31:0] > ((SwitchInfo:RandomFDBCap-1) / 16)[b.]
	AM[31:0] > 3071	AM[31:0] > 3071
MulticastForwarding-Table[a.]	AM[31:28] > (NodeInfo:NumPorts/16)[b.]	AM[31:28] > (NodeInfo:NumPorts/16)[b.]
	AM[8:0] > ((SwitchInfo:MulticastFDBCap-1) / 32)[b.]	AM[8:0] > ((SwitchInfo:MulticastFDBCap-1) / 32)[b.]
SMInfo[e]	For Set(), AM[31:0]!= 0x1,0x2,0x3,0x4,0x5 OR invalid SMInfo state transitions	For Set(), AM[31:0]!= 0x1,0x2,0x3,0x4,0x5 OR invalid SMInfo state transitions
	For Get(), AM[31:0]!= 0x0	For Get(), AM[31:0]!= 0x0
VendorDiag	AM[31:0] > 0x0000_FFFF	AM[31:0] > 0x0000_FFFF
LedInfo	AM[31:0] > 0x0	AM[31:0] > 0x0
Vendor Specific attributes with AttributeID ranging from 0xFF00 to 0xFFFF	AM[31:0] > 0x0000_FFFF	AM[31:0] > 0x0000_FFFF

a. For switch only.
b. Integer division with fraction truncated.
c. For CAs or routers, when AM is non-zero and not matching the port number where the SMP is received, all attribute components become RO. In such case, no status 7 due to illegal attribute components is possible.

d. Up to the implementation of choice. Using SwitchInfo:LinearFDBCap allows updates to the LinearForwardingTable beyond the SwitchInfo:LinearFDBTop before moving the SwitchInfo:LinearFDBTop.
e. For SM only.

14.2.6.6 SMP ATTRIBUTE COMPONENT ERRORS (STATUS_FIELD[4:2] = 0X7)

The Following tables list SMP attribute components that are of concern to the SM attribute component errors. These errors can only be detected while performing SubnSet() operations targeting RW components. Some RO components are also listed because of their special characteristics.

Table 165 Notice Attribute Component Errors

Component	Invalid Set() Component Value
NoticeToggle	no invalids, error handling statements exist for non matching NoticeToggle
NoticeCount	no invalids, error handling statements exist for NoticeCount > Notice queue depth

Table 166 NodeDescription Attribute Component Errors

All RO components, no component errors.

Table 167 NodeInfo Attribute Component Errors

All RO components, no component errors.

Table 168 SwitchInfo Attribute Component Errors

Component	Invalid Set() Component Value for Switch
LinearFDBTop	≥ SwitchInfo:LinearFDBCap
DefaultPort	no invalids
DefaultMulticastPrimaryPort	no invalids
DefaultMulticastNotPrimaryPort	no invalids
LifeTimeValue	no invalids
PortStateChange	no invalids

Table 169 GUIDInfo Attribute Component Errors

Component	Invalid Set() Component Value
GUID	no invalids

Table 170 PortInfo (CA/Router/Switch Port0) Attribute Component Errors

Component	Invalid Set() Component Value for BSP0	Invalid Set() Component Value for CA/Router/ESP0
M_Key	no invalids	no invalids
GidPrefix	no invalids	no invalids
LID	= 0x0000	= 0x0000
	≥ 0xC000	≥ 0xC000
MasterSMLID	= 0x0000	= 0x0000
	≥ 0xC000	≥ 0xC000
M_KeyLeasePeriod	no invalids	no invalids
LinkWidthEnabled (LWE)	N/A	1x PORT: 0x2, 0x4 ≤ LWE ≤ 0x8, 0xA, 0xC ≤ LWE ≤ 0xFE
		4x PORT: 0x4 ≤ LWE ≤ 0x8, 0xC ≤ LWE ≤ 0xFE
		12x PORT: 0x4 ≤ LWE ≤ 0x7, 0xC ≤ LWE ≤ 0xFE
PortState	N/A	0x2, ≥ 0x5
		set to Arm while in Arm, or Down
		set to Active while in Active, Initialize, or Down
PortPhysicalState	N/A	0x3[a], ≥ 0x4
LinkDownDefautState	N/A	≥ 0x3
M_KeyProtectBits	no invalids	no invalids
LMC	≥ 0x1	no invalids
LinkSpeedEnabled (LSE)	N/A	0x2 ≤ LSE ≤ 0xE

Table 170 PortInfo (CA/Router/Switch Port0) Attribute Component Errors (Continued)

Component	Invalid Set() Component Value for BSP0	Invalid Set() Component Value for CA/Router/ESP0
NeighborMTU	N/A	> PortInfo:MTUCap
		= 0
		> 5
MasterSMSL	no invalids	no invalids
VLHighLimit	N/A	no invalids
InitTypeReply	no invalids	no invalids
HOQLife	N/A	no invalids[b]
OperationalVLs	N/A	> PortInfo:VLCap
		> 5
M_KeyViolations	no invalids	no invalids
P_KeyViolations	no invalids	no invalids
Q_KeyViolations	no invalids	no invalids
SubnetTimeOut	no invalids	no invalids
LocalPhyErrors	N/A	no invalids
OverrunErrors	N/A	no invalids

a. Invalid only for the enhanced switch port 0.
b. For router only.

Table 171 PortInfo (Switch External Port) Attribute Component Errors

Component	Invalid Set() Component Value
LinkWidthEnabled (LWE)	1x PORT: 0x2, 0x4 ≤ LWE ≤ 0x8, 0xA, 0xC ≤ LWE ≤ 0xFE
	4x PORT: 0x4 ≤ LWE ≤ 0x8, 0xC ≤ LWE ≤ 0xFE
	12x PORT: 0x4 ≤ LWE ≤ 0x7, 0xC ≤ LWE ≤ 0xFE
PortState	0x2, ≥ 0x5
	set to Arm while in Arm, or Down
	set to Active while in Active, Initialize, or Down
PortPhysicalState	≥ 0x4

Table 171 PortInfo (Switch External Port) Attribute Component Errors (Continued)

Component	Invalid Set() Component Value
LinkDownDefautState	≥ 0x3
LMC	N/A
LinkSpeedEnabled (LSE)	0x2 ≤ LSE ≤ 0xE
NeighborMTU	> PortInfo:MTUCap
	= 0
	> 5
VLHighLimit	no invalids
VLStallCount	no invalids
HOQLife	no invalids
OperationalVLs	> PortInfo:VLCap
	> 5
PartionEnforcementIn-bound	set to 1 when SwitchInfo:InboundEnforcement-Cap is 0
PartionEnforcementOut-bound	set to 1 when SwitchInfo:OutboundEnforcement-Cap is 0
FilterRawInbound	set to 1 when SwitchInfo:FilterRawInboundCap is 0
FilterRawOutbound	set to 1 when SwitchInfo:FilterRawOutboundCap is 0
LocalPhyErrors	no invalids
OverrunErrors	no invalids

Table 172 P_Key Attribute Component Errors

Component	Invalid Set() Component Value
MembershipType	no invalids
P_KeyBase	no invalids

Table 173 SLtoVLMappingTable Attribute Component Errors

Component	Invalid Set() Component Value
SL0toVL	no invalids
SL1toVL	no invalids
SL2toVL	no invalids
SL3toVL	no invalids
SL4toVL	no invalids
SL5toVL	no invalids
SL6toVL	no invalids
SL7toVL	no invalids
SL8toVL	no invalids
SL9toVL	no invalids
SL10toVL	no invalids
SL11toVL	no invalids
SL12toVL	no invalids
SL13toVL	no invalids
SL14toVL	no invalids
SL15toVL	no invalids

Table 174 VLArbitrationTable Attribute Component Errors

Component	Invalid Set() Component Value
VL	no invalids
Weight	no invalids

Table 175 LinearForwardingTable Attribute Component Errors

Component	Invalid Set() Component Value
Port	> NodeInfo:NumPorts, read back as 0xFF No error status!

Table 176 RandomForwardingTable Attribute Component Errors

Component	Invalid Set() Component Value
LID	no invalids
Valid	no invalids
LMC	no invalids
Port	> NodeInfo:NumPorts when Valid = 0x1 No error status!

Table 177 Multicast ForwardingTable Attribute Component Errors

Component	Invalid Set() Component Value
PortMask	PortMask bits corresponding to ports that are beyond NodeInfo:NumPorts are read back as 0x0 No error status!

Table 178 SMInfo Attribute Component Errors

All RO components, no component errors.

Table 179 VendorDiag Attribute Component Errors

All RO components, no component errors.

Table 180 LedInfo Attribute Component Errors

Component	Invalid Set() Component Value
LedMask	no invalids

Table 181 Vendor Specific Attribute

No component errors.

14.3 SUBNET MANAGEMENT AGENT

Each CA and router, and switch will have a Subnet Management Agent (SMA) that communicates with the SMI and SM as described in 13.3.2 Required Managers and Agents on page 716. The SMA will respond and generate SMPs as described in Table 122 SM MAD Sources and Destinations on page 756. This section describes the detailed requirements of SMA behavior where the operations defined below assume the receipt of

a valid SMP. A SMP is valid if it satisfies all applicable validation checks as specified in 13.5.3 MAD Validation on page 755.

14.3.1 SUBNGET()

A SMA may receive a SMP from the subnet containing a SubnGet() at any time. The requester, the master SM, will fill the MADHeader:M_Key field of the SMP header with a M_Key that matches the value of the M_Key of the port corresponding to the receiving SMA if it expects the receiving SMA to check it.

A SMP containing a SubnGetResp() is returned according to the rules in 14.3.3 SubnGetResp() on page 853.

14.3.2 SUBNSET()

An SMA may receive a SMP from the subnet containing a SubnSet() at any time. The requester, the master SM, will fill the MADHeader:M_Key field of the SMP header with a M_Key that matches the value of the M_Key of the port corresponding to the receiving SMA if it expects the receiving SMA to check it.

C14-25: If the PortInfo:M_Key component is zero, the SMA **shall** update the appropriate components with the contents of the attribute contained in the SMP.

C14-26: If the PortInfo:M_Key component is non-zero and M_Key matching, if required, is successful according to the rules specified in 14.2.4 Management Key on page 806, the SMA **shall** update the appropriate components with the contents of the attribute contained in the SMP.

C14-27: The SMA **shall** ignore requests to change non-settable (RO) components of attributes.

A SMP containing a SubnGetResp() is returned according to the rules in 14.3.3 SubnGetResp() on page 853.

14.3.3 SUBNGETRESP()

C14-28: When the SMA receives a validated SubnGet() or SubnSet() and the PortInfo:M_Key component is zero, then the SMA **shall** generate a SubnGetResp().

C14-29: When the SMA receives a validated SubnGet() or SubnSet() and the PortInfo:M_Key component is non-zero and M_Key matching, if required, is successful according to the rules specified in 14.2.4 Management Key on page 806, then the SMA **shall** generate a SubnGetResp(), otherwise the request is silently discarded.

C14-30: If the SMA generates a SubnGetResp(), it **shall** fill the attribute identified in the request with the appropriate contents of component information.

C14-31: If the SMA generates a SubnGetResp(), it **shall** use the MADHeader:TransactionID obtained from the request SMP in the response SMP.

C14-32: This compliance statement is obsolete and has been removed.

If the SMA generates a SubnGetResp(), the content of MADHeader:M_Key in the SMP header is undefined. It could, for example, be copied without change from the request or zeroed out. If the SMA generates a SubnGetResp(), it should send the SMP containing the SubnGetResp() in less than PortInfo:RespTimeValue of the receiving port, where requirements for response time are described in 13.4.6.2 Timers and Timeouts on page 727.

After transmission of the response, the SMA discards any residual state associated with that SMP.

14.3.4 SUBNTRAP()

Traps may be issued by any port on the subnet. Ports that support this mechanism will indicate this by setting the PortInfo:CapabilityMask:IsTrapSupported bit.

o14-1: If the SMA generates a SubnTrap(), it **shall** fill the M_Key field of the SMP with zero.

o14-2: If the SMA generates a sequence of traps, the interval between successive traps **shall** not be smaller than the subnet timeout, which is specified by the PortInfo:SubnetTimeOut component.

This mechanism is used to limit the number of traps sent on the subnet.

o14-3: IThis compliance statement is obsolete and has been replaced by o14-3.2.1:.

o14-3.2.1: If the SMA generates a trap, it **shall** send it only in the following circumstances:

- if the SMA resides on a CA or router, PortInfo:PortState is Active for the port on which the trap is to be sent
- if the SMA resides on a switch with an Extended Switch Port 0, PortInfo:PortState is Active for Extended Switch Port 0

- if the SMA resides on a switch with Basic Switch Port 0, PortInfo:PortState is Active for the external switch port on which the trap will be sent.

o14-3.a1: If the SMA generates a trap, it **shall** set the source LID to the PortInfo:LID of the originating port.

This section describes the application of the architected traps for subnet management event reporting. The entire list of subnet management class traps are described in 14.2.5.1 Notices and Traps on page 812.

14.3.5 SUBNTRAPREPRESS()

An SMA may receive a SMP from the subnet containing a SubnTrapRepress() in reaction to a SubnTrap() sent by the SMA itself. The requester, the master SM, will fill the MADHeader:M_Key field of the SMP header with a M_Key that matches the value of the M_Key of the port corresponding to the receiving SMA if it expects the receiving SMA to check it.

o14-3.a2: If the PortInfo:M_Key component is zero, the SMA **shall** process the SubnTrapRepress() according to 13.4.9 Traps on page 741.

o14-3.a3: If the PortInfo:M_Key component is non-zero and M_Key matching, if required, is successful according to the rules specified in 14.2.4 Management Key on page 806, the SMA **shall** process the SubnTrapRepress() according to 13.4.9 Traps on page 741.

o14-3.a4: The SMA **shall** not send any message in response to a valid SubnTrapRepress() message.

14.3.6 PORT STATE CHANGE

Switches are capable of reporting port state changes.

o14-4: This compliance statement is obsolete and has been removed.

o14-5: This compliance statement is obsolete and has been replaced by o14-5.1.1:.

o14-5.1.1: If a switch supports Traps (PortInfo:CapabilityMask.IsTrapSupported is one), its SMA shall send trap 128 to the SM indicated by the PortInfo:MasterSMLID under any condition that would cause SwitchInfo:PortStateChange to be set to one. (See 14.2.5.4 SwitchInfo on page 819.)

o14-6: This compliance statement is obsolete and has been replaced by o14-6.1.1:.

o14-6.1.1: If a switch supports Notices (PortInfo:CapabilityMask.IsNoticeSupported is one), its SMA shall log a Notice using Notice:TrapNumber

128 under any condition that would cause SwitchInfo:PortStateChange to be set to one. (See 14.2.5.4 SwitchInfo on page 819.)

The contents of the trap or notice is filled with information from Table 133 Notice DataDetails For Trap 128 on page 814.

14.3.7 P_KEY MISMATCH ON SWITCH EXTERNAL PORTS

P_Key mismatch happens when a P_Key residing in the headers of an incoming or outgoing packet on a switch external port does not match any entry of the P_KeyTable for that switch external port as described in 18.2.4 Packet Relay Requirements on page 1044.

o14-6.1.2: If the switch management port's PortInfo:CapabilityMask.IsPKeySwitchExternalPortTrapSupported is set, the SMA on the switch management port **shall** monitor P_Key mismatches on each switch external port according to 18.2.4 Packet Relay Requirements on page 1044.

o14-6.1.3: If the switch management port's PortInfo:CapabilityMask.IsPKeySwitchExternalPortTrapSupported and PortInfo:CapabilityMask.IsTrapSupported are set, and if a P_Key mismatch is detected according to 18.2.4 Packet Relay Requirements on page 1044, then the SMA **shall** send a trap 259 to the SM indicated by the PortInfo:MasterSMLID. The DataDetails of the trap shall be filled with information from Table 139 Notice DataDetails For Trap 259 on page 817 for P_Key mismatches on a switch external port.

o14-6.1.4: If the switch management port's PortInfo:CapabilityMask.IsPKeySwitchExternalPortTrapSupported and PortInfo:CapabilityMask.IsNoticeSupported are set, and if a P_Key mismatch is detected according to 18.2.4 Packet Relay Requirements on page 1044, then the SMA **shall** log a Notice using Notice:TrapNumber 259. The DataDetails of the Notice shall be filled with information from Table 139 Notice DataDetails For Trap 259 on page 817 for P_Key mismatches on a switch external port.

14.3.8 TRANSPORT KEY MISMATCH

Transport key mismatch happens when a key residing in the headers of an incoming packet does not match the key for the destination QP during packet validation as described in 9.6 Packet Transport Header Validation on page 269.

C14-33: The SMA **shall** monitor P_Key and Q_Key mismatches detected by the transport services on that port.

C14-34: If a P_Key or Q_Key mismatch occurs, the SMA **shall** report the current count via the contents of PortInfo:P_KeyViolations or Port-

Info:Q_KeyViolations components of the PortInfo attribute (see 14.2.5.6 PortInfo on page 821).

o14-7: If the port supports Traps as indicated in the PortInfo:CapabilityMask.IsTrapSupported, the SMA **shall** send a trap 257 or 258 to the SM indicated by the PortInfo:MasterSMLID for P_Key and Q_Key mismatches, respectively.

o14-8: If the port supports Notices as indicated the PortInfo:CapabilityMask.IsNoticeSupported, the SMA **shall** log a notice for P_Key and Q_Key mismatches using Notice:TrapNumber 257 or 258.

The contents of the trap or notice is filled with information from Table 138 Notice DataDetails For Traps 257 and 258 on page 816 for P_Key and Q_Key mismatches, respectively.

14.3.9 M_KEY MISMATCH

As a result of the M_Key residing in the SMP header, the SMA is responsible for checking it. The SMA will perform an M_Key check as described in 14.2.4.1 Levels of Protection on page 807. If the check fails and a lease period countdown is not already in effect, the SMA starts a lease period countdown as described in 14.2.4.2 Lease Period on page 807.

o14-9: If a port receives a SMP for which an M_Key mismatch is detected, then, if the port supports Traps as indicated in PortInfo:CapabilityMask.IsTrapSupported, the SMA of that port **shall** send a trap 256 to the SM indicated by PortInfo:MasterSMLID.

o14-10: If a port receives a SMP for which an M_Key mismatch is detected, then, if the port supports Notices as indicated PortInfo:CapabilityMask.IsNoticeSupported, the SMA of that port **shall** log a Notice using Notice:TrapNumber 256.

The contents of the trap or notice is filled with information from Table 137 Notice DataDetails For Trap 256 on page 815.

14.3.10 LINK LAYER ERRORS

The link layer performs error detection and recovery as described in 7.12 Error detection and handling on page 219. The SMA is responsible for monitoring the Local link integrity, excessive buffer overrun, and flow control update errors detected by the link layer of the port.

o14-11: When a Local Link Integrity error occurs on a port that supports Traps (PortInfo:CapabilityMask.IsTrapSupported is set), the port's SMA **shall** send Trap 129 to the SM at the address contained in PortInfo:MasterSMLID; when an Excessive Buffer Overrun error occurs, Trap 130

shall be sent; and when a Flow Control Update error occurs, Trap 131 **shall** be sent.

o14-12: If the port supports Notices as indicated the PortInfo:CapabilityMask.IsNoticeSupported, the SMA **shall** log a notice using Notice:TrapNumber 129, 130, or 121, respectively, when the Local link integrity, excessive buffer overrun, or flow control update counters increment.

The contents of the trap or notice is filled with information from Table 134 Notice DataDetails For Traps 129, 130 and 131 on page 814 for Local link integrity, excessive buffer overrun, and flow control update counter changes, respectively.

14.3.11 CHANGE CAPABILITYMASK

The PortInfo:CapabilityMask component is RO and therefore cannot be changed by the SM. However, the PortInfo:CapabilityMask may be modified at runtime under normal subnet operational conditions by entities on an endnode. The SMA is responsible for monitoring and reporting changes in the PortInfo:CapabilityMask.

o14-12.1.1: If the port supports Traps as indicated in the PortInfo:CapabilityMask.IsTrapSupported and the port is capable of sending the CapabilityMask trap as indicated in the PortInfo:CapabilityMask.IsCapabilityMaskNoticeSupported bit, then the SMA **shall** send a trap 144 to the SM indicated by the PortInfo:MasterSMLID when the PortInfo:CapabilityMask is modified at runtime.

o14-12.1.2: If the management port supports Notices as indicated in the PortInfo:CapabilityMask.IsNoticeSupported and the port is capable of logging the CapabilityMask notice as indicated in the PortInfo:CapabilityMask.IsCapabilityMaskNoticeSupported, the SMA **shall** log a Notice using Notice:TrapNumber 144 when the PortInfo:CapabilityMask is modified at runtime.

The contents of the trap or notice is filled with information from Table 135 Notice DataDetails For Trap 144 on page 815 for CapabilityMask changes.

14.3.12 CHANGE SYSTEMIMAGEGUID

The NodeInfo:SystemImageGUID component is RO and therefore cannot be changed by the SM. However, it may be modified at runtime under normal subnet operational conditions by entities on an endnode. The SMA is responsible for monitoring and reporting changes in the NodeInfo:SystemImageGUID.

o14-12.1.3: If the port supports Traps and SystemImageGUID (as indicated by PortInfo:CapabilityMask.IsTrapSupported and PortInfo:Capabil-

ityMask.IsSystemImageGUIDSupported) then the SMA shall send a trap 145 to the SM indicated by the PortInfo:MasterSMLID when the NodeInfo:SystemImageGUID is modified at runtime.

o14-12.1.4: If the management port supports Notices and SystemImageGUID (as indicated by PortInfo:CapabilityMask.IsNoticeSupported and PortInfo:CapabilityMask.IsSystemImageGUIDSupported) then the SMA shall log a Notice using Notice:TrapNumber 145 when the NodeInfo:SystemImageGUID is modified at runtime.

The contents of the trap or notice is filled with information from Table 136 Notice DataDetails For Trap 145 on page 815 for SystemImageGUID changes.

14.4 SUBNET MANAGER

There may be one or more Subnet Managers operating on a subnet as described in 13.3.2 Required Managers and Agents on page 716. There may be several SMs on a particular node, each residing on different subnets.

C14-35: An SM **shall** always be associated with one port and one subnet.

C14-35.1.1: A Subnet Manager (SM) **shall** indicate its presence on the subnet by setting the IsSM bit in the PortInfo:CapabilityMask on the port where it resides (see Table 145 PortInfo on page 822).

The mechanism for activating an SM is beyond the scope of the specification. The ability to modify the IsSM bit is provided by the Verbs for HCAs (see 11.2.1.3 Modify HCA Attributes on page 556) and is otherwise beyond the scope of the specification.

Each SM is always in a particular state: Master, Standby, Discovering or Not-active.

The algorithm used to initialize the subnet, the algorithm for adding/deleting routes in response to subnet changes, the mechanisms for failover from master SM to standby SM, and the mechanism for transfer of mastership from master SM to standby SM are beyond the scope of the specification. However, there are mechanisms specified in this section that may be used to support these operations.

C14-36: An SM **shall** comply with the state machine shown in Figure 194 SMInfo State Transitions on page 861 during its startup and **shall** become either a master or standby SM.

Correct execution of the state machine ensures that there be only one Master SM on a subnet at any time and that after startup, a SM becomes either a Standby or Master on the subnet.

Furthermore, the state machine specifies how a single Master SM is maintained during subnet topology changes, packet loss, addition/removal of SMs, and subnet mergers. Subsequent sections include the specification of optional mechanism that may be used by SMs to communicate and a description of some SM operations on the subnet, but none of these are required for SM compliance.

14.4.1 SM STATE MACHINE

The behavior of the SM is specified in terms of the SM state machine. This section starts by defining the specific mechanisms used by the SM: the SMInfo attribute, control packets that SMs may exchange, a set of timers, and the exception conditions reported to the higher layer (administrator).

Each SM provides a SMInfo attribute that is specified in Table 158 SMInfo on page 840 and is exported from the port where it resides.

C14-37: This compliance statement is obsolete and has been replaced by C14-37.1.1:

C14-37.1.1: The SMInfo:Priority and SMInfo:SM_Key **shall** be configurable only through an out-of-band mechanism that is outside the scope of this specification.

C14-37.1.2: The SM **shall** keep the SMInfo:Priority provided to it (see C14-37.1.1:) in nonvolatile memory.

The contents of the components in the SMInfo attribute determine which SM in a multi-SM subnet becomes Master: the one with the highest Priority and the lowest GUID.

Each Standby SM should be ready to become Master when the current Master fails (or gets disconnected). Also, mastership will be handed over when the Master detects another SM with a higher Priority (or same Priority and lower GUID), e.g., during merger of two subnets. Handover takes place only between SMs that have the right SM_Key.

Under certain circumstances, e.g., when the number of Standby SMs becomes an obstacle to scaleability, then a Master SM may force other SMs to become Not-active.

C14-38: In order to assure interoperability, each SM **shall** respond to SubnGet(SMInfo) or SubnSet(SMInfo) with a SubnGetResp(SMInfo).

Figure 194 SMInfo State Transitions on page 861 summarizes the states that a SM may represent in the SMInfo:SMState.

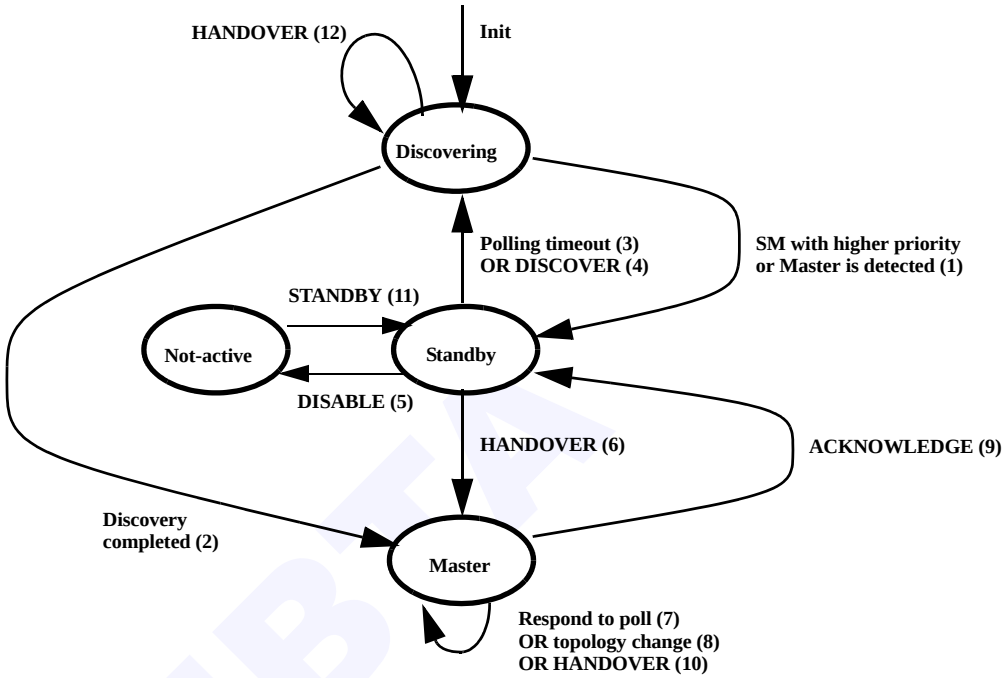


Figure 194 SMInfo State Transitions

The state transitions correspond to the event numbers in the text below. The following sections describe the behavior of the SM in each of the states and the (externally driven) events that cause state changes.

14.4.1.1 CONTROL PACKETS

Control packets may be exchanged between SMs using a SMP that contains a SubnSet(SMInfo) where the MADHeader:AttributeModifier is used to select from one of the following actions specified in Table 182 SM Control Packets on page 862. The SM is not required to generate these control packets and may use mechanisms that are beyond the scope of the specification to implement similar functions, however, a SM is required to correctly respond to them.

Table 182 SM Control Packets

MADHeader:AttributeModifier	Description
1	HANDOVER: Is used to initiate the process of handing over Mastership to a higher priority Standby SM or Master.
2	ACKNOWLEDGE: Is used to acknowledge the handover
3	DISABLE: Is used to disable a Standby SM.
4	STANDBY: Is used to return a Not-active SM to Standby.
5	DISCOVER: Causes a Standby SM to go to Discovering.

C14-38.1.1: An SM that receives a SubnSet(SMInfo) control packet requesting an invalid state transition **shall** ignore the requested action and return a SubnGetResp(SMInfo) with a MADHeader:Status code of 7.

Invalid state transition control packets include those with a MADHeader:AttributeModifier of 1, 3, 4, or 5 that were not sent by the Master SM; or those with a MADHeader:AttributeModifier of 2 that were not sent by a Standby SM; or those requesting a state transition that is not present in Figure 194 SMInfo State Transitions on page 861. When an invalid state transition control packet is received by an SM, that SM may also want to notify a higher layer (through an interface beyond the scope of the specification).

14.4.1.2 DISCOVERING STATE

DISCOVERING is the initial state.

C14-39: At startup, a SM **shall** enter the DISCOVERING state.

C14-40: In the DISCOVERING state, the SM **shall** perform repetitive SubnGet(*) to find all nodes and SMs on the subnet.

Section 14.4.2 Subnet Discovery Actions on page 867 summaries many of the attributes that are collected during discovery. The SM will typically use direct-routed SMPs to reach all the endnodes. The sequence of discovery is implementation specific and beyond the scope of the specification.

C14-41: This compliance statement is obsolete and has been replaced by C14-41.1.1:

C14-41.1.1: An SM in the DISCOVERING state **shall** yield and change its SMInfo:SMState to STANDBY if it finds another SM that either:

- has SMInfo:SMState = MASTER;

- or is in a state other than NOT-ACTIVE and has either a higher Priority than its own or the same Priority and a lower GUID.

See Figure 194 SMInfo State Transitions on page 861, number 1. At this point the SM stops the discovery and starts operating as a Standby SM.

C14-42: This compliance statement is obsolete and has been replaced by C14-41.1.1:

C14-42.1.1: An SM in the DISCOVERING state **shall** assume the role of a Master by changing its SMInfo:State to MASTER if it completes the discovery process without finding a Master or a higher priority (lower GUID) SM that is in a state other than NOT-ACTIVE.

C14-43: The master SM **shall** initially send to all the nodes on the subnet SubnSet(PortInfo) SMPs with PortInfo:MasterSMLID and PortInfo:MasterSMSL that specify a path to itself.

See Figure 194 SMInfo State Transitions on page 861, number 2.

C14-44: If the SM discovers that it does not have a M_Key required to configure a CA, switch, or router on the subnet it **shall** notify the higher-layer (through an interface beyond the scope of the specification).

14.4.1.3 STANDBY STATE

C14-45: Standby SMs **shall** not configure the subnet.

C14-46: Each Standby SM **shall** poll the Master SM with SubnGet(SMInfo) SMPs, addressed to its PortInfo:MasterSMLID. As long as the Standby determines that the Master is alive, it stays in SMInfo:SMState = STANDBY.

The minimum interval between polling is set by the higher-layer (through an interface beyond the scope of the specification). The actual interval may be longer for Standby SMs with lower Priority or when there is a larger number of Standby SMs on the subnet. The actual polling interval is installation specific and is not specified in the architecture. The Master may use the optional control packets to disable Standby SMs if it determines that there is excessive polling in the subnet.

C14-47: If the Standby SM does not receive a SubnGetResp(SMInfo) that indicates progress in the ActCount, within the number of retries that is set by the higher-layer (through an interface beyond the scope of the specification), then it should conclude that the Master is no longer alive (or accessible) and it **shall** change its SMInfo:SMState back to DISCOVERING.

See Figure 194 SMInfo State Transitions on page 861, number 3.

C14-48: If a Standby SM receives a DISCOVER packet, i.e. a SubnSet(SMInfo) with MADHeader:AttributeModifier set to the value of 5, then it **shall** change its SMInfo:SMState to DISCOVERING.

See Figure 194 SMInfo State Transitions on page 861, number 4.

C14-49: If a Standby SM receives a DISABLE packet, i.e., a SubnSet(SMInfo) with MADHeader:AttributeModifier set to the value of 3, then it **shall** change its SMInfo:SMState to NOT-ACTIVE.

See Figure 194 SMInfo State Transitions on page 861, number 5. This allows the Master to disable Standby SMs if it determines that the amount of polling creates a scaleability problem.

Event 6 specifies the Standby SM behavior when a Master SM determines that the Standby SM has higher priority and the correct SM_Key, and subsequently relinquishes mastership to that Standby SM. The Master's behavior is specified in 14.4.1.5 Master State on page 865.

C14-50: If a Standby SM receives a HANDOVER control packet, i.e., a SubnSet(SMInfo) with MADHeader:AttributeModifier set to the value of 1, it **shall** return a SubnGetResp(SMInfo).

The steps necessary to take control of the subnet are beyond the scope of specification. However, the standby SM may, for example, perform the following operations:

1) The standby SM receiving the HANDOVER control packet obtains necessary topology information, possibly obtaining data defined as required record attributes specified in 15.2.5.1 Summary of Attributes on page 888 from subnet administration residing with the current Master SM.
2) The standby SM should send to all the nodes on the subnet a SubnSet(PortInfo) with MasterSMLID and MasterSMSL that specify a path to itself.
3) The standby SM may send the current Master SM an ACKNOWLEDGE control packet, i.e. an SubnSet(SMInfo) with MADHeader:AttributeModifier set to the value of 2.
4) If the standby SM does not receive a SubnGetResp(SMInfo), it should notify the higher-layer (through an interface beyond the scope of the specification). This is an indication that the Master may have died in the middle of an unsuccessful hand over.
5) After receiving a SubnGetResp(SMInfo), it assumes the role of a Master by changing its SMInfo:State to MASTER. See Figure 194 SMInfo State Transitions on page 861, number 6.

14.4.1.4 NOT-ACTIVE STATE

C14-51: If a SM is in the NOT-ACTIVE state, it **shall** indicate this by setting the SMInfo:SMState to NOT-ACTIVE.

C14-52: If the SM is in the NOT-ACTIVE state, it **shall** not send SubnSet() or SubnGet() SMPs.

C14-53: If the SM is in the NOT-ACTIVE state, it **shall** respond to SubnSet(SMInfo) and SubnGet(SMInfo) SMPs.

C14-54: This compliance statement is obsolete and has been replaced by C14-54.1.1:.

C14-54.1.1: If the SM is in the NOT-ACTIVE state and it receives a STANDBY packet, i.e., a SubnSet(SMInfo) with MADHeader:AttributeModifier set to the value of 4, it **shall** change its state to STANDBY.

See Figure 194 SMInfo State Transitions on page 861, number 11.

14.4.1.5 MASTER STATE

The Master starts its operation by topology discovery, LID verification and assignment (if applicable), path verification and calculation, etc. (as specified in 14.4.3 Initialization Actions on page 868).

C14-55: Only the Master SM **shall** configure subnet nodes.

C14-56: This compliance statement is obsolete and has been deleted.

C14-57: If the M_Key protection mechanism, as described in 14.2.4.1 Levels of Protection on page 807, is being used, the Master SM **shall** sweep the subnet at a rate that will refresh the lease period of every port on the subnet.

Section 14.4.6 Subnet Sweeping on page 878 describes the sweep activities.

C14-58: The Master **shall** increment the SMInfo.ActCount every time it performs a management operation or issues an SMP.

When the SM in Master state receives a valid SubnGet(SMInfo) or SubnSet(SMInfo), it should respond with a SubnGetResp(SMInfo) when M_Key matching, if required, is successful as described in 14.4.7 Authentication on page 878. This is required in order to support the Standby polling mechanism.

C14-59: If during the sweep the Master detects a topology change, then it **shall** perform the operations listed below:

- If the change is a link going down, then the Master needs to possibly establish new paths and send new MasterSMLID/SLs to the affected nodes. The details are beyond the scope of the specification.
- If the Master detects a new link, then it starts discovering the subnet beyond the new links, using (partially) direct routed SMPs.

  If the SM discovers that it does not have a M_Key required to configure a CA, switch, or router on the subnet, it will notify the higher-layer (through an interface beyond the scope of the specification).

C14-60: This compliance statement is obsolete and has been replaced by C14-60.2.1:.

C14-60.2.1: If a Master SM finds another Master SM with lower priority (or same priority and higher GUID) it **shall** ensure that it is the highest priority (or same priority and lower GUID) on the subnet, and if so it **shall** wait for the other Master (or Masters) to relinquish control if its portion of the subnet.

C14-61: This compliance statement is obsolete and has been replaced by C14-61.1.1:

C14-61.1.1: If the Master SM finds an SM that is in a state other than NOT-ACTIVE, has a higher priority (or same priority and lower GUID), and has the appropriate SM_Key; then it **shall** complete the discovery or sweep in order to determine the highest priority SM (with an appropriate SM_Key) in the new part of the subnet (if applicable) and it **shall** relinquish control of its portion of the subnet to that SM.

The steps necessary to transfer control of the subnet from one master SM to another is beyond the scope of specification, however, the master SM may use the optional control packets to perform the handover process as follows:

- It may complete operations in progress.
- It sends the higher priority SM a HANDOVER packet, i.e. a SubnSet(SMInfo) with MADHeader:AttributeModifier set to the value of 1.
- It continues responding to polls from Standby SMs until it receives an ACKNOWLEDGE packet, i.e., a SubnSet(SMInfo) with MADHeader:AttributeModifier set to the value of 2 from the higher priority SM.
- When it receives an ACKNOWLEDGE packet, it will change its SMInfo:SMState to STANDBY and return a SubnGetResp(SMInfo). See Figure 194 SMInfo State Transitions on page 861, number 9.

- If it does not receive an ACKNOWLEDGE packet, then it informs the higher-layer (through an interface beyond the scope of the specification).

C14-61.2.1: If a Master SM determines that a higher priority Master SM does not have the proper SM_Key, then it **shall** not relinquish mastership of its portion of the subnet, and **shall** not change the state of the subnet.

C14-61.2.2: If a Master SM determines that a lower priority Master SM has not performed a handover within a vendor-specific time period, then it **shall** not change the state of the subnet.

Should a Master SM determine the occurrence of the conditions described in either C14-61.2.1: or C14-61.2.2:, it should report the occurrence to a higher level protocol (through an interface beyond the scope of the specification). An acceptable technique for determining whether or not a Master SM will ever do a HANDOVER can include polling of the Master SM as would normally be done by a standby SM (see C14-46:).

C14-61.2.3: If a SM in the MASTER state receives a HANDOVER control packet, i.e., a SubnSet(SMInfo) with MADHeader:AttributeModifier set to the value of 1, it shall return a SubnGetResp(SMInfo). See Figure 194 SMInfo State Transitions on page 861, number 10.

14.4.2 SUBNET DISCOVERY ACTIONS

The SM collects information from the attributes and records them for later use during configuration of the subnet. The discovery algorithm is outside the scope of the specification, however, discovery may consist of:

- probing the subnet with directed route packets
- loading a topology database from persistent storage
- a combination of information that is loaded from persistent storage and obtained by probing subnet nodes

During discovery, the SM scans the attributes described in 14.2.5 Attributes on page 809 to obtain information not limited to the following:

- VLs on each Port
- MTU of the Port
- Link Width on each Port
- Link Speed on each Port
- Physical topology, connectivity of links between nodes
- P_Key table sizes
- GUID table sizes
- support for various capabilities

- device type: switch, CA, or router
- power-on diagnostic status
- for switches,
  - size of switch linear-forwarding or random-forwarding tables
  - support for multicast forwarding table and size
  - presence of the optional VL arbitration table
  - presence of the optional SL-to-VL mapping table

14.4.3 INITIALIZATION ACTIONS

The algorithms and policies that are necessary to set many of the subnet attributes are outside the scope of the specification. However, there is a core set of attributes that the SM is responsible for setting in order to make the subnet functional.

C14-62: This compliance statement is obsolete and has been replaced by C14-62.1.1:.

C14-62.1.1: The Master SM **shall** initialize the subnet components specified in the following Table 183 Initialization on page 868.

Table 183 Initialization

Component	Description
PortInfo:LID	The SM **shall** assign a unicast LID address to each endport on the subnet. LID usage is described in 4.1.2 Channel Adapter, Switch, and Router Addressing Rules on page 147 and 4.1.3 Local Identifiers on page 147. The SM **shall** also set up the forwarding tables of the switches in the subnet such that each base LID of a port on a CA, Router, or switch port 0 is reachable from any other port on the subnet within the boundaries set by partitions.
PortInfo:LMC	The SM **shall** assign an LMC for each CA and router port on the subnet. LMC usage is described in 4.1.3 Local Identifiers on page 147. The SM may program the LMC on a port to any value between 0 and 7 to allow use of multiple LIDs in addressing the port. The SM **shall** not assign overlapping ranges of LIDs based on LMCs to different ports.
PortInfo:GidPrefix	The SM **shall** assign a Subnet Prefix for the subnet based on the presence of a router and the rules specified in 4.1.3 Local Identifiers on page 147.
PortInfo:OperationalVL	The SM **shall** initialize the VL tables for CAs, switches and routers. The SM will examine the supported VLs in the PortInfo:VLCap at both ends of every link and sets the maximum number of VLs by setting the PortInfo:OperationalVL at each end to the smaller of the two supported number of VLs. The description of VL initialization resides in 7.6.7 Initialization and Configuration on page 188. In the case of an enhanced switch port 0, there is, effectively, just one end of the link; in that case OperationalVLs may be set to any value allowing any number of VLs up to and including the VLCap of the port.

Table 183 Initialization (Continued)

Component	Description
PortInfo:NeighborMTU	The SM **shall** initialize the port MTU for CAs, switches and routers. • The SM on other than an enhanced switch port 0 **shall** examines the supported MTU size in the PortInfo:MTUCap at both ends of every link and set the maximum MTU parameter on the ports in their PortInfo.NeighborMTU at each end to the smaller of the two supported sizes. • In the case of an enhanced switch port 0, there is, effectively, just one end of the link; in that case NeighborMTU **shall** be set to any value up to and including the MTUCap of the port.
PortInfo:SubnetTimeOut	The SM **shall** set the maximum trap generation rate for all nodes in the subnet by initializing the PortInfo.SubnetTimeOut component in all ports as described in 13.4.6.2.1 PortInfo:SubnetTimeout on page 727.
PortInfo:MasterSMLID	The SM **shall** store the LID of the port where it resides in the PortInfo:MasterSMLID of each port on the subnet.
PortInfo:MasterSMSL	The SM **shall** store the SL required for sending a non-SMP message to the SM using that LID in the PortInfo:MasterSMSL of each port on the subnet
PortInfo:PortPhysicalState	The default state on power-on is polling as described in Volume 2.
PortInfo:LinkDownDefault-State	The default state on power-on is polling as described in Volume 2.
PortInfo:VLHighLimit	The SM **shall** set the Limit of High-Priority limit for the number of bytes of high-priority packets that can be transmitted if the ports on both ends of a link may be operated with multiple data VLs as described in 7.6 Virtual Lanes Mechanisms on page 180
PortInfo:M_Key	The SM may initialize the PortInfo:M_Key for each port on the subnet as described in 14.2.4.5 Initialization on page 809. The rules for assigning these values is outside the scope of the specification.
PortInfo:M_KeyProtectBits	The SM may initialize the PortInfo:M_KeyProtectBits for each port on the subnet as described in 14.2.4 Management Key on page 806. The rules for assigning these values is outside the scope of the specification.
PortInfo:M_KeyLeasePeriod	The SM may initialize the PortInfo:M_KeyLeasePeriod for each port on the subnet as described in 14.2.4 Management Key on page 806. The rules for assigning these values is outside the scope of the specification.
PortInfo:M_KeyViolations	The SM **shall** clear the PortInfo:M_KeyViolations component for all ports on the subnet.
PortInfo:P_KeyViolations	The SM **shall** clear the PortInfo:P_KeyViolations component for all ports on the subnet.
PortInfo:Q_KeyViolations	The SM **shall** clear the PortInfo:Q_KeyViolations component for all ports on the subnet.
PortInfo:VLStallCount	The SM **shall** set a value for the PortInfo:VLStallCount as described in 18.2.5.4 Transmitter Queueing on page 1057. The rules for assigning these values is outside the scope of the specification.

Table 183 Initialization (Continued)

Component	Description
PortInfo:HOQLife	The SM **shall** set a value for the PortInfo:HOQLife as described in 18.2.5.4 Transmitter Queueing on page 1057. The rules for assigning these values is outside the scope of the specification.
PortInfo:DiagCode	The SM may check the PortInfo:DiagCode of every port on the subnet. The rules for correcting faults detected on ports is outside the scope of the specification.
PortInfo:InitTypeReply	If the SM implements Reinitialization (SA's ClassPortInfo:IsReinitSupported = 1), it **shall** set these bits in PortInfo in accordance with the initialization performed prior to activating a port. See 14.4.4 Node Reinitialization on page 871. Otherwise, these bits **shall** be set to 0.
GUIDInfo	The SM may assign GUIDs to ports to form GIDs as described in 4.1.1 GID Usage and Properties on page 143. There is one pre-assigned read-only GUID for each port that has the same value as NodeInfo:PortGUID. The requirements for setting additional GUIDs are beyond the scope of the specification.
SwitchInfo:LinearFDBTop	On a switch that supports a linear forwarding table, the SM will program the highest LID to port mapping used as described in 14.2.5.4 SwitchInfo on page 819.
SwitchInfo:DefaultPort	On a switch that supports a random forwarding table, the SM **shall** set the default port as described in 18.2.4.3.2 Random Forwarding Table Requirements on page 1051. The rules for assigning these values are outside the scope of the specification.
SwitchInfo:DefaultMulticastPrimaryPort	The SM **shall** set the *DefaultMulticastPrimaryPort* as described in 18.2.4.3.3 Required Multicast Relay on page 1053. The rules for assigning these values are outside the scope of the specification.
SwitchInfo:DefaultMulticastNotPrimaryPort	The SM **shall** set the *DefaultMulticastNotPrimaryPort* as described in 18.2.4.3.3 Required Multicast Relay on page 1053. The rules for assigning these values are outside the scope of the specification.
SwitchInfo:LifeTimeValue	The SM **shall** set a value for the LifeTimeValue as described in 18.2.5.4 Transmitter Queueing on page 1057. The rules for assigning these values are outside the scope of the specification.
VLArbitrationTable	VL arbitration described in 7.6.9 VL Arbitration and Prioritization on page 188 **shall** be set by the SM for the output link of each CA, switch, and router.
SLtoVLmappingTable	The application of VL is described in 7.6.6 VL Mapping Within a Subnet on page 186. The SM will initialize the SL-to-VL mapping tables. The rules for assigning these values are outside the scope of the specification. The SM **shall** check for the existence of the SLtoVLmappingTable and initializes it, if present.
P_KeyTable	The SM may initialize the P_Key table by setting entries in the P_KeyTable attribute for ports. It may also enable P_Key checking in switches. The policy for assigning P_Keys is in general outside the scope of the specification. However, the SM **shall** ensure that one of the P_KeyTable entries in every node contains either the value 0xFFFF (the default P_Key, full membership) or the value 0x7FFF (the default P_Key, partial membership). The purpose of this specific P_Key value is to provide communication with Subnet Administration (see 15.4.2 Locating Subnet Administration on page 923).

Table 183 Initialization (Continued)

Component	Description
LinearForwardingTable	Unicast forwarding tables will be set by the SM based on route policy decisions and Switch capabilities. The SM **shall** setup LID-to-port mappings if the Switch supports a Linear Forwarding Table as indicated by the SwitchInfo:LinearFDBCap component.
RandomForwardingTable	If the Switch supports a Random Forwarding Table as indicated by the SwitchInfo:RandomFDBCap component, the SM **shall** set up LID/LMC range to port mappings based on route policy decisions.
MulticastForwardingTable	If a Switch supports a Multicast Forwarding Table as indicated by the SwitchInfo:MulticastFDBCap component, the SM may setup LID to multi-port mappings in the Multicast Forwarding Table based on route policy decisions.

C14-62.1.2: When establishing the contents of switch forwarding tables and SL to VL maps, the subnet manager **shall** ensure that no cyclic flow control dependencies exist in the fabric.

Cyclic dependencies in flow control can cause deadlock and subsequent failure of an IBA fabric. There are a number of routing methods that may be employed to prevent these dependencies. These include pruned and fat tree structures, dimension order routing in meshes and hyper cubes, and use of multiple virtual lanes to break the flow control cycle in routing loops. While IBA does not specify a particular routing method, whatever method is utilized must ensure deadlock-free operation.

C14-62.1.3: The SM **shall** ensure that no packet entering a subnet at any port with any combination of SL and DLID can cause subnet deadlock.

This can be accomplished even if the subnet manager utilizes virtual lanes to break cyclic flow control dependencies by putting appropriate entries in the switches' SL to VL mapping tables to ensure that improper use of SLs by nodes originating traffic (for example, nodes utilizing an SL inconsistent with path records provided by the subnet administrator) are discarded.

The above compliance statement does not imply that the subnet is required to deliver packets with improper SLs. It is permissible to discard such packets.

C14-62.1.4: From every endport within the subnet, the SM **shall** provide at least one reversible path to every other endport. See 13.5.4 Response Generation and Reversible Paths on page 768.

14.4.4 NODE REINITIALIZATION

Any of the types of nodes in an IBA network may require reinitialization, meaning reloading of attributes set by the SM as part of subnet initialization. This may occur for a variety of reasons involving failure of microcode,

software, and hardware. This is true for any type of node: HCA, TCA, switch, or router; and it may extend beyond the node itself to data held by the SM and/or SA about the node, such as the node's subscriptions to events, membership in multicast groups, and service records identifying services on the failed node.

The degree of reinitialization required optimally depends on the circumstances of the fault. For example, the following types of distinct errors could occur:

1) Supervisor failure, for example, failure of the operating system, hypervisor, or switch/router microcode. In this case, the supervisor will presumably reboot and reinitialize all information internal to the node. (The SM has no ability to control this.) This case is effectively equivalent to a hot unplug of the port, followed by hot plugging it. All SM/SA data, like subscriptions, should be removed in this case. There is benefit to reloading the port's attributes with data similar to what it previously contained, but that benefit is not overwhelming; for example, using the same LID previously used may eliminate reprogramming of some switch tables.
2) A channel adapter failure exhibiting itself to the supervisor as, for example, a failure of verbs to perform as expected; but without any failure of the supervisor. In this situation, there may be significant added value an SM vendor can provide in being able to reload the port with exactly the same data it contained previously, and in not eliminating any SM/SA stored data. If that is done and the CA restarted quickly, while connections to other ports will likely have to be recreated, the supervisor may not have to terminate and restart applications, a potentially very lengthy and disruptive operation.
3) A port failure exhibiting itself only in that a port's SMA becomes unresponsive to the SM: MADs sent to the SMA do not return responses or return anomalous responses. The CA may otherwise be operating normally, but the SM has little choice but to force down the link to that port (by forcing down a switch port attached to it), since it has lost control of that port. Like case 2, there may be great value in reinitializing the port exactly as it was previously, if the supervisor can take advantage of that fact to avoid disruption.
4) The link attached to a port goes down due to an error that cannot be corrected transparently by automatic retraining, such as the loss of a physical lane—for example, a 4x link is reduced to a 1x link. In this case the value loaded by the SM will not change (LinkWidthSupported still says 4x, so the SM will set LinkWidthEnabled to either allow 1x and 4x or just 1x as it wishes), but the LinkWidthActive will come up as 1x at most, causing probable widespread changes in PathMTUs elsewhere.

Of course, combinations of any the above could occur simultaneously.

The handling of all such cases begins when the SM is signalled that some form of reinitialization is required by the port going “down,” i.e., its PortInfo:PortState goes to the Down state. This can occur naturally (e.g., case 4), be forced by a supervisor using the 11.2.1.3 Modify HCA Attributes on page 556 or equivalent TCA function to signal case 1 or case 2; or be forced by the SM itself in case 3 by writing the Down value to the PortInfo:PortState. When the SM does not force the link down, the SM can detect it either in the course of a subnet scan or by receiving trap 128, see Table 131 Traps on page 812.

Subsequent actions depend on whether the SM, the port, or both implement the Reinitialization option as indicated by the SA’s ClassPortInfo:CapabilityMask.IsReinitSupported (for the SM; see 15.2.1.3 SA-Specific ClassPortInfo:CapabilityMask Bits on page 884) and the port’s PortInfo:CapabilityMask.IsReinitSupported (for the port; see 14.2.5.6 PortInfo on page 821).

If the SM does not support reinitialization, the action taken is vendor-specific.

If both the SM and the port support Reinitialization, the SM then determines the type of initialization desired by inspecting the PortInfo:InitType component, set by the supervisor by using the 11.2.1.3 Modify HCA Attributes on page 556 or equivalent TCA function. The bits of this field represent requests by the supervisor to not reload any data (NoLoad), preserve the prior content of the port’s attributes (PreserveContent), and/or preserve subscriptions and all other SM/SA data referencing that port (PreservePresence).

o14-12.1.5: If the SM supports Reinitialization, and a port does not, then the SM **shall** reinitialize a port that has entered the down state as if the port did support Reinitialization and had set all its PortInfo:InitType bits to 0; and **shall** set all InitTypeReply bits to 0.

It may not, however, be possible for the SM to honor one or more of these requests. For example, the SM may not have kept any record of the prior content of that port’s attributes; whether it does so is vendor value-add. Even if the SM retains prior loaded data, it may be impossible to reload with exactly the same data; for example, the values of P_Keys loaded into the port’s P_Key Table may have changed since the port was last active; or the LID it was assigned may have been reused for another port. The PortInfo:InitTypeReply component enables the SM to signal the supervisor which, if any, of its requests were honored.

o14-12.1.6: If both the SM and a port to be reinitialized support Reinitialization, and the SM did successfully perform an action requested by any of the PortInfo:InitType component bits, the SM **shall** set the corresponding PortInfo:InitTypeReply bit to 1 (except for DoNotResuscitate,

which has no corresponding reply bit; see below). Otherwise the SM shall set the corresponding reply bit to 0 (except for DoNotResuscitate; see below).

It is possible that the most convenient thing for the SM to do is to reload the port with attribute data it previously contained, even though the port did not request this explicitly by setting PortInfo:InitType.PreserveContent to 1.

o14-12.1.7: If both the SM and a port to be reinitialized support Reinitialization, and the SM did reload data identical to the data most recently loaded into the port's attributes, then the SM **shall** set PortInfo:InitType.PreserveContentReply to 1. The SM **shall not** set any other PortInfo:InitType reply bit to 1 unless the corresponding PortInfo:InitType request bit was 1.

The use of PreservePresence, in particular, has the potentially valuable effect but has a restriction and an additional implication. Its value is in allowing a short CA failure to be less disruptive to operation of a port. Connections made to that port will probably be broken and must be re-established; but other state data like reservations, service record registrations, etc., can remain in force and their possible cached status in other nodes is not affected.

However, PreservePresence should not be honored by the SM if it is unable to also reinitialize the port with exactly the same data it had previously (i.e., if it was unable to honor NoLoad or PreserveContent). Doing so means that data cached in other places may not, in fact, be valid when the port resumes operation.

o14-12.1.8: If both the SM and a port to be reinitialized support Reinitialization, and the SM has reloaded the port with data not known to be the same as the data previously (e.g., it could not honor NoLoad or PreserveContent requests by a port), the SM **shall not** honor a PreservePresence request by that port.

Additionally, honoring PreservePresence has the additional implication that Report()s for trap number 65, indicating a port is out of service, not be issued. If those Report()s were issued, other nodes holding data related to this port, such as managers holding subscriptions to traps, would in all likelihood eliminate that data since they've been informed that the node is no longer reachable.

o14-12.1.9: If both the SM and a port to be reinitialized support Reinitialization, and the SM was able to honor a PreservePresence request, Report()s for trap 65 (and 64) **shall not** be issued.

Otherwise, the definition of the trap number 64 event (see 14.4.9 In and Out of Service Traps on page 880) implies that a link going down should cause Report()s of trap 65 be sent for the port attached to that link; and when the port is reinitialized, Report()s of trap 64 should be sent.

PreservePresence has no effect on the ServiceLease processing of ServiceRecords by SA. I.e., even if a PreservePresence request was honored, ServiceRecord entries in SA may have been deleted because their ServiceLease period expired while a port was down.

PortInfo:InitType also provides the DoNotResuscitate bit. This exists to eliminate a possible race between the SM, reading the PortInfo:InitType bits; and the supervisor, setting them. DoNotResuscitate requests the SM to delay reinitializing the port until this bit is set to 0. Setting this bit to 1 immediately following port activation, and leaving it set to 1 while operating, should (in the absence of some other failures) guarantee that the SM will see that bit set to 1 when the SM first accesses the port after a link down. As a result, the SM will not preemptively reinitialize the port while the supervisor is assessing the situation, figuring out how it wants the other initialization bits set. DoNotResuscitate being 1 also tells the SM that trap 65 (out of service) should not be immediately Report()ed, since PreservePresence may be requested; and that SM/SA resources used by this port should not be recovered, for the same reason. After this bit becomes zero, the settings of PortInfo:InitType may or may not cause trap 65 to be reported and resources to be reclaimed. There is no corresponding reply bit for DoNotResuscitate, since a node is notified resuscitation by an event (see 11.6.3 Asynchronous Events on page 637). As was true of the other PortInfo:InitType bits, the SM may not honor a DoNotResuscitate request, and it likely should not do so after some (vendor-specific) time has passed. For example, the supervisor could have failed in a manner that left the node in a coma but the DoNotResuscitate bit reporting “1” forever.

Table 184 PortInfo:InitType Interpretations on page 876 shows the interpretation of all possible combinations of PortInfo:InitType bits. In that table, the phrase “preserve the port’s external presence” refers to pre-

serving all SM/SA data related to that port and refraining from sending Report()s of trap numbers 64 and 65.

Table 184 PortInfo:InitType Interpretations

InitType Bits				Interpretation	
NoLoad	Preserve-Content	Preserve-Presence	DoNot-Resuscitate	Send trap # 64/65 report & release SM/SA resources?	Description
any	any	any	1	wait	Port is requesting that the SM delay reinitialization while DoNotResuscitate is 1, including the possible Report()ing of trap #s 64 and 65 and the releasing of SM/SA resources associated with this port.
0	0	0	0	Y	Port is oblivious to reinitialization issues; SM reinitializes it, and may do so in any way that is convenient.
0	0	1	0	N if content preserved; otherwise Y	Port does not request that all data be the same as the most recent content loaded by the SM; but if it is (by choice of the SM), then the port's external presence should be preserved.
0	1	0	0	Y	Port is requesting that it be reloaded and all data be the same as the most recent content loaded by the SM; but its external presence should not be maintained.
0	1	1	0	N if content preserved; otherwise Y.	Port is requesting that it be reloaded and all data be the same as the most recent content loaded by the SM and that its external presence should be maintained.
1	0	0	0	Y	Port is requesting that no data be loaded into its attributes at all, asserting that the last-loaded data still exists and is valid; but if it is loaded, SM may do so in any way that is convenient.
1	0	1	0	N if content preserved; otherwise Y.	Port is requesting that no data be loaded into its attributes; but if this request is not honored, SM may do so in any way that is convenient.
1	1	0	0	Y	Port is requesting that no data be loaded into its attributes; if this request is not honored, port requests that all data be the same as the most recent content loaded by the SM.
1	1	1	0	N if content preserved; otherwise Y.	Port is requesting that no data be loaded into its attributes; if this request is not honored, port requests that all data be the same as the most recent content loaded by the SM.

Finally, it should be noted that reinitialization is in many circumstances a work around for a problem that ultimately must be diagnosed and corrected. Since reinitialization may itself destroy information needed for diagnosis, it is important to maintain adequate logging facilities that record reasons for reinitialization problems. For example: Did a supervisor force a port down because the supervisor ran into internal problems, or did it do so because it noticed some form of problem with the other side of the link?

Without a supervisor-maintained log of the reason for the forced downing of the link, it may be very difficult to separate those cases after the fact.

14.4.5 PORT STATE TRANSITIONS

When power is applied to a device, its ports attempt to reach an operational state according to the steps described in the InfiniBand Architecture specification, Volume 2, Link/Phy Interface Chapter and 6.2 Services provided by the Physical Layer. on page 163. A physical subnet is established when a group of devices are connected together and the state of a set of ports reaches operational state.

C14-63: A SM **shall** determine that a subnet is operational when the PortInfo:Portstate on the port where it resides is at the initialize state.

The SM may access the management entities of remote CAs, switches, and routers while the ports along the physical links are in initialize state since the SMI on that port will recognize a packet on QP0 and VL15, with a LID destination address 0xFFFF as referring to the SMA.

The SM may change the state of a port to active, armed, initialize or down that are described in 14.4.5 Port State Transitions on page 877.

The SM may perform most port and device configuration activities while the PortInfo:Portstate is in the initialize state. However, all control and configuration options are also available in the armed state and the active state. In addition to the link level behaviors, the PortInfo:Portstate has an additional role because it is manipulated by the SM to communicate to endnodes the readiness of the subnet. CAs and Routers may start sending packets on the subnet if one of its ports enters the active state. As a result, moving a port from active state is likely to be disruptive to subnet activity.

An SM that becomes the master SM may enable transmission of packets through the subnet at any time. This is accomplished after establishing routes by setting the switch forwarding tables and initializing the other attributes as described in 14.4.3 Initialization Actions on page 868 for CAs, switches, and routers along those routes, and then setting the PortInfo:Portstate to armed for the ports along those routes. The SM changes the state of an Endnode from armed to active to signal to the Endnode that it may begin to send packets. Ports on switches and along that route and endnodes that are destinations of those packets will transition from armed to active automatically as described in 14.4.5 Port State Transitions on page 877.

The SM may reset port related state by:

1) setting the PortInfo:LinkDownDefaultState to *polling* state

2) setting the PortInfo:Portstate to the down state.

The PortInfo:Portstate should return to the initialize state after clearing its state as described by the link state machine in Figure 50 Link State Machine on page 170.

14.4.6 SUBNET SWEEPING

C14-64: After the subnet is up and running, the SM **shall** periodically gather information about topology changes, PortInfo:CapabilityMask changes, and Notices reported by nodes.

This is referred to as sweeping the subnet. The frequency of subnet sweeps is undefined for this architecture, as it will vary due to topology and other implementation considerations.

The SM detects topology changes by examining the port state of nodes in the subnet. For example, when the value of the PortInfo.Portstate component of a port changes from down to initialize, the SM will use directed routed packets to probe the other end of the link on that port to determine what has been added to the subnet. Conversely, if the PortInfo.Portstate component changes from active to down, the SM may perform operations such as updating switch forwarding tables to delete routes to the end-node(s) that are no longer accessible. To speed up detection of port state changes, switches support a SwitchInfo:PortStateChange component, described in Table 142 SwitchInfo on page 819, that the SM may examine. If the state of this component indicates that the state of one of the switch ports has changed, the SM may proceed to check the status of each port on that switch.

14.4.7 AUTHENTICATION

During initialization of a SMP, the SM may fill in the MADHeader:M_Key field of the SMP with the value that matches the M_Key stored in the destination port if it expects the destination management entity to check it.

C14-65: The SM **shall** not check the MADHeader:M_Key stored in a SubnGetResp(*).

C14-66: If the SM receives a validated SMP containing a SubnSet(SMInfo) or SubnGet(SMInfo) and the PortInfo:M_Key component is zero, then the SM **shall** generate a SubnGetResp.

C14-67: If the SM receives a validated SMP containing a SubnSet(SMInfo) or SubnGet(SMInfo), and the PortInfo:M_Key component is non-zero, and M_Key matching, if required, is successful according to the rules specified in 14.2.4 Management Key on page 806, then the SM **shall** generate a SubnGetResp(). Otherwise the SubnSet(SMInfo) or SubnGet(SMInfo) is silently discarded.

C14-68: When a Master SM receives a SMP containing a SubnTrap(), it **shall** not check that the MADHeader:M_Key field matches the PortInfo:M_Key of the port where the SMP was received.

The SMInfo:SM_Key is used by the Master SM to authenticate other standby SMs and master SMs, in the case of a subnet merge, on the subnet. Exactly how the key is used is implementation specific. A SM should fill the SM_Key in the SMInfo:SM_Key component in a response if it expects the requesting SM to check it.

14.4.8 SM DISABLE MECHANISM

C14-69: If a SM can reside on a port, a vendor defined, out-of-band mechanism **shall** be provided that when asserted will disable the capability of running a SM from that port and the state of the mechanism **shall** be indicated in the Portinfo:CapabilityMask.IsSMdisabled bit.

C14-70: When the Portinfo:CapabilityMask.IsSMdisabled bit is asserted, the port behavior **shall** be:

- SubnSet(SMInfo) or SubnGet(SMInfo) sent to that port **shall** be discarded
- SubnSet(*) or SubnGet(*) **shall** not be sent from that port
- The Portinfo:CapabilityMask.IsSM bit for that port **shall** not be set.

C14-71: When PortInfo:CapabilityMask.IsSMdisabled is not asserted, the port behavior **shall** be:

- SubnSet(SMInfo) or SubnGet(SMInfo) sent to that port will be forwarded to management entities if the appropriate entity is operational
- SubnSet(*) or SubnGet(*) may be sent by management entities from the port
- The Portinfo:CapabilityMask.IsSM bit is controlled by management entities behind that port.

C14-72: The state of the PortInfo:CapabilityMask.IsSMdisabled on a port **shall** be changeable at any time while the port is operational.

The mechanism for changing the state of the Portinfo:CapabilityMask.IsSMdisabled bit is beyond the scope of the specification.

Changing the state of Portinfo:CapabilityMask.IsSMdisabled bit from asserted to not-asserted while the port is otherwise operational allows assertion of the *IsSM* bit in the PortInfo:CapabilityMask and allows activation of an SM behind the port (see 14.4 Subnet Manager on page 859).

14.4.9 IN AND OUT OF SERVICE TRAPS

Trap numbers 64 and 65, indicating when an endport comes in or out of service (see Table 131 Traps on page 812), are never used in SubnTrap() SMPs sent on a subnet; the corresponding events are generated within the Master SM itself, and so do not have to be signalled to the SM by SubnTrap() messages. These trap numbers are only used to allow the SM to report the events to endnodes through the Subnet Administrator (see 15.4.3 Event Forwarding Subsystem on page 923) using SubnAdmReport() GMPs. They may also appear in the corresponding SubnAdmReportResp() GMPs.

The events reported by these Notice()s indicate that an endport's reachability from a subscribing endport has changed. Endport A is reachable from subscriber endport B if there is a PathRecord with source B and destination A (see 15.2.5.16 PathRecord on page 899). Endport reachability may change as a result of changes to restrictions on access through partitioning, or endports commencing or ceasing participation on the subnet. These events are a logical equivalent of hot plug and unplug as seen from the viewpoint of the subscriber.

C14-72.1.1: When an endport A becomes reachable from an endport B that has subscribed to trap 64, the SM **shall** cause the SA to send a SubnAdmReport() to B using trap number 64 with the associated Notice attribute DataDetails providing the GID of A as described in Table 132 Notice DataDetails For Traps 64, 65, 66, and 67 on page 814

C14-72.1.2: When an endport A ceases to be reachable from an endport B that has subscribed to trap 65, the SM **shall** cause the SA to send a SubnAdmReport() to B using trap number 65 with the associated Notice attribute DataDetails providing the GID of A as described in Table 132 Notice DataDetails For Traps 64, 65, 66, and 67 on page 814.

14.4.10 MULTICAST GROUP CREATE/DELETE TRAPS

Trap numbers 66 and 67, indicating when a multicast group is created or deleted, are events that are generated by the SA when a multicast group is created or deleted (see 15.2.5.17 MCMemberRecord on page 908). These trap numbers are used to report these events through the SA using SubnAdmReport() GMPs. These traps are similar to trap numbers 64/65 (see 14.4.9 In and Out of Service Traps on page 880) in that they are never used in SubnTrap() SMPs sent on a subnet. These traps are optionally supported by the SA as indicated by the SA's ClassPortInfo:CapabilityMask.IsUDMulticastSupported bit.

o14-12.1.10: If SA supports UD multicast, then if an endport has subscribed to trap 66 for a specific or wildcarded MGID, the SA **shall** send a SubnAdmReport() to that endport when a multicast group is created with a matching MGID (as specified in 15.2.5.17 MCMemberRecord on page

908) using trap 66 with the associated Notice attribute DataDetails as described in Table 132 Notice DataDetails For Traps 64, 65, 66, and 67 on page 814.

o14-12.1.11: If SA supports UD multicast, then if an endport has subscribed to trap 67 for a specific or wildcarded MGID, the SA **shall** send a SubnAdmReport() to that endport when the last receiving member of multicast group with a matching MGID leaves the group (as specified in 15.2.5.17 MCMemberRecord on page 908) using trap 67 with the associated Notice attribute DataDetails as described in Table 132 Notice DataDetails For Traps 64, 65, 66, and 67 on page 814.

14.4.11 CLIENT REREGISTRATION

Client reregistration allows the Subnet Manager to request that a client reregister all subscriptions previously requested from this port. The SM may request this at any time of any port supporting this option. A reason for the SM doing this might be that the SM suffered a failure and as a result lost its own records of such subscriptions.

The SM class uses the PortInfo attribute to affect client reregistration. A port indicates it supports client reregistration for the SM class by setting PortInfo:CapabilityMask.IsClientReregistrationSupported = 1.

o14-12.2.1: If a port supports client reregistration (PortInfo.IsClientReregistrationSupported = 1), the SMA **shall** respond to a SubnSet(PortInfo) with PortInfo:ClientReregister=1 as follows:

- a SubnGetResp(PortInfo) **shall** be returned with PortInfo:ClientReregister =1
- an asynchronous unaffiliated event of type Client Reregistration **shall** be generated (see 11.6.3.3 Unaffiliated Asynchronous Events on page 640).

o14-12.2.2: Compliance statement o14-12.2.1: is the only situation in which any SMA **shall** return a PortInfo with ClientReregister=1; in all other cases, it **shall** be 0.

CHAPTER 15: SUBNET ADMINISTRATION

15.1 INTRODUCTION AND OVERVIEW

All compliance statements in this chapter from releases prior to 1.1 are obsolete and have been deleted from the specification.

This chapter defines IBA Subnet Administration (SA) and its functions: the MADs used, and the functions with which they are associated.

C15-0.1.1: Every IBA subnet **shall** provide SA.

15.1.1 SA FUNCTION

Through the use of Subnet Administration class MADs, SA provides access to and storage of information of several types, some optional.

C15-0.1.2: The information that **shall** be provided by SA is specified in Table 189 Subnet Administration Attributes (Summary) on page 888.

The types of information involved are:

- Information that endnodes require for operation in a subnet. Such information includes paths between endnodes, notification of events, service attributes, etc. This information is required.
- Information that is non-algorithmic, typically. Information that cannot be recovered algorithmically by inspection of the network after a power-on or initialization event. Such information includes partitioning data, M_Keys, SL to VL mappings, etc. This information is required to allow off-line migration from one vendor's subnet management implementation to another's. This is required.
- Information that may be useful to other management entities such as standby SMs, who may, for example, wish to use it to maintain synchronization with the master SM. Such information includes subnet topology data, switch forwarding tables, etc. This is optional.

In order to perform these activities, SA includes two functions:

- A query subsystem required to identify the information to be sent and received
- An event-forwarding subsystem that forwards SM-received traps and notices to subscribed parties.

The actual SA implementation is outside the scope of architecture. The actual access QP and DLID may be redirected by the GSI.

15.1.2 RELATIONSHIP BETWEEN SA AND THE SM

Much, but not all, of the information provided by SA is created or collected by the SM. SA must therefore have a close relationship with the master SM. That relationship is defined as follows:

- SA is part of the SM. Its functions are discussed separately from the SM only for convenience of description. This descriptive convenience is not intended to imply or require any particular implementation organization of the SM (or SA) by any vendor.
- As is the case for any class of IB management, SA functions may be implemented on a node separate from the one holding the SM; whether this is done is vendor-specific. If any SM function is implemented at a location different from the one identified as holding the SM, including but not limited to SA functions, any or all communication between that function and any other SM elements is vendor-specific.

C15-0.1.3: Should an SM be elected master SM, all of its components **shall** also be implicitly elected master, including but not limited to SA, however they may be implemented. If an SM ceases to be master, all of its components, including but not limited to SA, all **shall** cease responding to messages from client nodes.

15.1.3 OVERVIEW

The remainder of this chapter first defines the MADs used by SA. It also defines the operation of SA. The SA operations described include locating SA and SA methods and their operation. Also described are identification of information attributes, access restrictions that must be implemented, and event forwarding.

15.2 SA MADS

This section defines the MADs sent and received by SA.

C15-0.1.4: The SA MADs are GMPs and **shall** conform to MAD use as specified in 13.4 Management Datagrams on page 717.

15.2.1 SA MAD FORMAT

C15-0.1.5: Subnet Administration **shall** use the datagram format shown in Figure 195 Subnet Administration Format on page 884 with the fields specified in Table 185 Subnet Administration Fields on page 884.

C15-0.1.6: The MADHeader:ClassVersion component for the SA class **shall** be 2 for this version of the specification.

15.2.1.1 SA HEADER

bytes	bits 31-24	bits 23-16	bits 15-8	bits 7-0
0-32	Standard MAD Header with RMPP header (see Figure 170 RMPP Header Layout on page 772)			
36	SM_Key			
40				
44	AttributeOffset		Reserved	
48	ComponentMask			
52				
56	SubnetAdminData (200 bytes)			
...				
252				

Figure 195 Subnet Administration Format

15.2.1.2 SA HEADER FIELDS

Table 185 Subnet Administration Fields

Field	Length	Description
SM_Key	64 bits	Subnet Manager verification key: Used to authenticate a request as being from a trusted source. • For SA requests this field **shall** be either zero or a valid SM_Key. If neither of these values is present, the request **shall** be discarded, and it is recommended that suitable note of this occurrence be made by the SM. • For SA responses, this field **shall** be set to zero. Refer to Chapter 14: Subnet Management on page 794 and 15.4.1 Restrictions on Access on page 921.
AttributeOffset	16 bits	In an RMPP DATA packet: Number of 8-byte words from the beginning of an attribute to the beginning of the next attribute. To compute the offset in bytes, multiply AttributeOffset by eight. AttributeOffset **shall not** change during any single RMPP transfer. Otherwise: Ignored
Reserved	16 bits	Reserved
ComponentMask	64 bits	Used to indicate attribute components to be used for SA operations. Bit 0 maps to first attribute component, bit 1 the second attribute component, and so forth. A bit set to one indicates attribute component is used to form a SA operation, otherwise field is to be ignored.Refer to 15.4.4 Administration Query Subsystem on page 923
SubnetAdminData	1600 bits	Data field where attribute content is stored.

15.2.1.3 SA-SPECIFIC CLASSPORTINFO:CAPABILITYMASK BITS

Table 186 SA-Specific ClassPortInfo:CapabilityMask Bits on page 885

specifies the use of class-specific bits of ClassPortInfo:CapabilityMask by the Subnet Administration class:

C15-0.1.7: Subnet Administration **shall** behave in accordance with the descriptions in Table 186 SA-Specific ClassPortInfo:CapabilityMask Bits on page 885.

Table 186 SA-Specific ClassPortInfo:CapabilityMask Bits

Name	Bit	Description
IsSubnetOptional-RecordsSupported	8	If this value is 1, SA **shall** support all attributes listed as optional in Table 189 Subnet Administration Attributes (Summary) on page 888, except for MCMemberRecord, TraceRecord, and MultiPathRecord. This bit **shall not** be used to indicate support a subset of those attributes. If this value is 0, SA **shall** support none of the above attributes and methods.
IsUDMulticastSup-ported	9	If this value is 1, SA **shall** support MCMemberRecord as listed in Table 189 Subnet Administration Attributes (Summary) on page 888.
IsMultiPathSupported	10	If this value is 1, SA **shall** support the TraceRecord and MultiPathRecord attributes as listed in Table 189 Subnet Administration Attributes (Summary) on page 888, as well as SubnAdmGetTraceTable(), SubnAdmGetMulti(), and SubnAdmGetMultiResp() methods, as listed in Table 187 Subnet Administration Methods on page 885.
IsReinitSupported	11	If this value is 1, reinitialization **shall** be supported by the SM/SA for this port, as described in 14.4.4 Node Reinitialization on page 871.

See 13.4.8.1 ClassPortInfo on page 734 for an overall description of ClassPortInfo:CapabilityMask.

15.2.2 SUMMARY OF METHODS

Table 187 Subnet Administration Methods on page 885 summarizes the methods provided by the Subnet Administration class. Several of these are common methods described in 13.4.5 Management Class Methods on page 721; some are unique to this class. Subnet Administration methods are described in more detail in 15.4 Operations on page 921.

C15-0.1.8: SA **shall** support all the methods listed in Table 187 Subnet Administration Methods on page 885. All Method Type Values not listed in the table are reserved.

Table 187 Subnet Administration Methods

Method Type	Value	Optional/ Required	Description
SubnAdmGet()	0x01	Required	Request a get (read) of an attribute from a node
SubnAdmGetResp()	0x81	Required	The response from an attribute get or set request
SubnAdmSet()	0x02	Required	Request a set (write) of an attribute in a node. The responder **shall** issue a SubnAdmGetResp() as its response.

Table 187 Subnet Administration Methods (Continued)

Method Type	Value	Optional/ Required	Description
SubnAdmReport()	0x06	Required	Forward an event previously subscribed for
SubnAdmReportResp()	0x86	Required	Reply to a SubAdmReport() method
SubnAdmGetTable()	0x12	Required	Table request
SubnAdmGetTableResp()	0x92	Required	Table request response
SubnAdmGetTraceTable()	0x13	Optional	Request path trace table
SubnAdmGetMulti()	0x14	Optional	Multi-packet request
SubnAdmGetMultiResp()	0x94	Optional	Multi-packet response
SubnAdmDelete()	0x15	Required	Request to delete an attribute
SubnAdmDeleteResp()	0x95	Required	Response to SubnAdmDelete() method

15.2.3 SUBNET ADMINISTRATION STATUS VALUES

For SA, values for the class-specific status bits (see 13.4.7 Status Field on page 731) have the meaning specified in Table 188 SA MAD Class-Specific Status Encodings on page 886.

Table 188 SA MAD Class-Specific Status Encodings

Name	Value	Meaning
NO_ERROR	0	No class-specific error
ERR_NO_RESOURCES	1	Insufficient resources to complete the request. This error may be returned for any SA request which could not be processed due to insufficient resources.
ERR_REQ_INVALID	2	Supplied request or update is invalid.
ERR_NO_RECORDS	3	No records match query. May be returned only by a SubnAdmGetResp(). Never returned for an RMPP response because an RMPP transaction with a payload length of zero is a valid transaction.
ERR_TOO_MANY_RECORDS	4	Too many records match query.
ERR_REQ_INVALID_GID	5	Invalid GID in SA request.
ERR_REQ_INSUFFICIENT_COMPONENTS	6	Request did not contain the required components.
Reserved	7-255	Reserved; reception of any of these status codes **shall** be considered an error.

15.2.4 ATTRIBUTES AND ATTRIBUTE TABLES

In common with all management classes, the AttributeID component of the standard MAD header identifies the format and semantics of the data in each MAD. The format and semantics of SA attributes are described in detail in section 15.2.5 Attributes on page 888.

C15-0.1.9: For every subnet node, port, and link discovered by the SM, SA **shall** make available all the attributes necessary to describe the state of that node, port, or link. The exact attributes available will depend upon the type of entity. If the SM discovers that a node, port, or link is no longer accessible or down, the attributes describing it **shall** no longer be available from SA.

There is a class of SA attributes which have characteristics different from attributes found in other management classes. In addition, unlike other classes, SA can return tables of attributes. This section describes those characteristics and the tables.

15.2.4.1 EMBEDDED ATTRIBUTES

A significant function of SA is to allow user programs and other entities to use GMPs to obtain data known by the SM. For this reason, in many cases the data formats and semantics used by SA are actually the formats and semantics defined by the SM; in effect, SM attribute data is often embedded in SA attributes.

The SM data embedded in an attribute may be stored in SA, or stored in the SM, or obtained on demand from the nodes which contain it, or otherwise obtained. Which is done for any attribute is implementation-dependent. The format and semantics of the embedded SM data is neither defined by SA nor described in this chapter, but rather defined and described by the SM; see 14.1 Subnet Management Model on page 794 for that information.

15.2.4.2 RECORD IDENTIFIER (RID) FIELDS

In addition to embedded data, SA attributes contain additional information which identifies the entity whose attribute is embedded. For example, PortInfoRecord contains, in addition to the SM-defined PortInfo data itself, the LID of the switch containing the port whose PortInfo is referred to, and the Port Number of that port on that switch. The fields containing that identifying information are referred to as the Record Identifier fields (RID fields). RID fields are always at the start of an attribute. The specific RID fields used for a particular attribute depend on the type of attribute. The individual attribute descriptions define each attribute's RID fields in detail.

The LIDs used in RID fields may in effect be aliased. This happens because use of the LMC facility for subnet multipathing (see 7.11 Subnet Multipathing on page 219) can map multiple LIDs to the same port. All

such aliased values are valid RID field values: Given a RID field containing a LID that can be used to communicate with a given port, SA must reference information about that port. This is true both for queries and for methods that write data. Attributes returned from a query, however, contain the base LID.

C15-0.1.10: Any LID usable to route a packet to a port may be used in a query as the LID component in a RID, and **shall** identify the port to which it routes.

C15-0.1.11: Query responses **shall** contain a port's base LID in any LID component of a RID.

15.2.4.3 TABLES

Collections of attributes transferred to or from subnet administration are called tables. These tables consist of a single type of attribute, such as NodeRecords.

15.2.5 ATTRIBUTES

This section first provides a summary of all the SA attributes, and then lists the data format of each, with descriptions where warranted.

15.2.5.1 SUMMARY OF ATTRIBUTES

C15-0.1.12: SA **shall** support the attributes and methods listed as required in Table 189 Subnet Administration Attributes (Summary) on page 888, and.Table 190 Subnet Administration Attribute / Method Map on page 890. respectively. All attribute IDs not listed in Table 189: Subnet Administration Attributes (Summary) are reserved.

Table 189 Subnet Administration Attributes (Summary)

Attribute Name	Attribute ID	Optional / Required	Embedded Attribute	Description
ClassPortInfo	0x0001	R	n	Class information. See 13.4.8.1 ClassPortInfo on page 734
Notice	0x0002	R	n	Notice information. See 13.4.8.2 Notice on page 737
InformInfo	0x0003	R	n	Subscription (Inform) Information. See 13.4.8.3 InformInfo on page 739
NodeRecord	0x0011	R	Y	Container for NodeInfo. See 15.2.5.2 NodeRecord on page 891
PortInfoRecord	0x0012	R	Y	Container for PortInfo. See 15.2.5.3 PortInfoRecord on page 891

Table 189 Subnet Administration Attributes (Summary) (Continued)

Attribute Name	Attribute ID	Optional / Required	Embedded Attribute	Description
SLtoVLMappingTableRecord	0x0013	R	Y	Container for SLtoVLMappingTable entry. See 15.2.5.4 SLtoVLMappingTableRecord on page 892
SwitchInfoRecord	0x0014	O	Y	Container for SwitchInfo. See 15.2.5.5 SwitchInfoRecord on page 892
LinearForwardingTableRecord	0x0015	O	Y	Container for LinearForwardingTable entry. See 15.2.5.6 LinearForwardingTableRecord on page 892
RandomForwardingTableRecord	0x0016	O	Y	Container for RandomForwardingTable entry. See 15.2.5.7 RandomForwardingTableRecord on page 893
MulticastForwardingTableRecord	0x0017	O	Y	Container for MulticastForwardingTable entry. See 15.2.5.8 MulticastForwardingTableRecord on page 893
SMInfoRecord	0x0018	O	Y	Container for SMInfo. See 15.2.5.10 SMInfoRecord on page 894
InformInfoRecord	0x00F3	O	Y	Container for InformInfo. See 15.2.5.12 InformInfoRecord on page 894
LinkRecord	0x0020	O	n	Inter-node linkage information. See 15.2.5.13 LinkRecord on page 895
GuidInfoRecord	0x0030	O	Y	Container for a port's GUIDInfo. See 15.2.5.18 GuidInfoRecord on page 916
ServiceRecord	0x0031	R	n	Information on advertised services. See 15.2.5.14 ServiceRecord on page 895
P_KeyTableRecord	0x0033	R	Y	Container for P_Key Table. See 15.2.5.11 P_KeyTableRecord on page 894
PathRecord	0x0035	R	n	Information on paths through the subnet. See 15.2.5.16 PathRecord on page 899
VLArbitrationTableRecord	0x0036	R	Y	Container for VLArbitrationTable entry. See 15.2.5.9 VLArbitrationTableRecord on page 893
MCMemberRecord	0x0038	O	n	Multicast member attribute. See 15.2.5.17 MCMemberRecord on page 908
TraceRecord	0x0039	O	n	Path trace information. See 15.2.5.19 TraceRecord on page 916
MultiPathRecord	0x003A	O	n	Request for multiple paths. See 15.2.5.20 MultiPathRecord on page 917
ServiceAssociationRecord	0x003B	O	n	ServiceRecord ServiceName/ServiceKey association. See 15.2.5.15 ServiceAssociationRecord on page 899

The AttributeModifier (see Table 112 Common MAD Fields on page 719) from the common header is not used with any of the Subnet Administration attributes.

Table 190 Subnet Administration Attribute / Method Map on page 890 associates SA attributes with methods.

Table 190 Subnet Administration Attribute / Method Map

Attribute	Get	Set	Report	GetTable	GetTraceTable	GetMulti	Delete
ClassPortInfo	X						
Notice			X				
InformInfo		X					
NodeRecord	X			X			
PortInfoRecord	X			X			
SLtoVLMapping-TableRecord	X			X			
SwitchInfoRecord	X			X			
LinearForwarding-TableRecord	X			X			
RandomForwarding-TableRecord	X			X			
MulticastForwarding-TableRecord	X			X			
VLArbitrationTableRecord	X			X			
SMInfoRecord	X			X			
InformInfoRecord	X			X			
LinkRecord	X			X			
GUIDInfoRecord	X			X			
ServiceRecord	X	X		X			X
P_KeyTableRecord	X			X			
PathRecord	X			X			
MCMemberRecord	X	X		X			X
TraceRecord[a]					X		
MultiPathRecord						X	
ServiceAssociationRecord	X			X			

a. TraceRecord is returned only in SubnAdmGetTableResp(), see 15.4.9 SubnAdmGetTraceTable(): Trace a Path on page 928

The detailed layouts of all the SA-specific attributes follows. Attributes which are containers for Subnet Management attributes require little description beyond their layout; others have more extensive descriptions

15.2.5.2 NODERECORD

Table 191 NodeRecord

Component		Length(bits)	Offset(bits)	Description
RID	LID	16	0	For a CA or router: LID of the port. For a switch: LID of switch port 0.
	Reserved	16	16	Reserved
NodeInfo		320	32	NodeInfo attribute contents; see Table 141 NodeInfo on page 818
NodeDescription		512	352	NodeDescription attribute contents; see Table 140 NodeDescription on page 818

There is one NodeRecord for each endport on a subnet.

Note: If a channel adapter or router has multiple ports on the same subnet, there will be multiple NodeRecords available for that node from SA, one for each possible PortGUID value of NodeInfo for that node on that subnet.

15.2.5.3 PORTINFORECORD

Table 192 PortInfoRecord

Component		Length(bits)	Offset(bits)	Description
RID	EndportLID	16	0	For a CA or router: LID of the port. For a switch: LID of switch port 0.
	PortNum	8	16	For a switch: port number For a channel adapter or router: reserved
	Reserved	8	24	Reserved
PortInfo		432	32	PortInfo attribute contents; see Table 145 PortInfo on page 822

15.2.5.4 SLTOVLMAPPINGTABLERECORD

Table 193 SLtoVLMappingTableRecord

Component		Length(bits)	Offset(bits)	Description
RID	LID	16	0	For a CA or router: LID of the port. For a switch: LID of switch port 0
	InputPortNum	8	16	For a switch: input port number For a channel adapter or router: reserved
	OutputPortNum	8	24	For a switch: output port number For a channel adapter or router: reserved
Reserved		32	32	Reserved
SLtoVLMappingTable		64	64	SLtoVLMappingTable attribute contents; see Table 149 SLtoVLMappingTable on page 835

15.2.5.5 SWITCHINFORECORD

Table 194 SwitchInfoRecord

Component		Length(bits)	Offset(bits)	Description
RID	LID	16	0	LID of switch port 0
	Reserved	16	16	Reserved
SwitchInfo		136	32	SwitchInfo attribute contents; see Table 142 SwitchInfo on page 819

15.2.5.6 LINEARFORWARDINGTABLERECORD

Table 195 LinearForwardingTableRecord

Component		Length(bits)	Offset(bits)	Description
RID	LID	16	0	LID of switch port 0
	BlockNum	16	16	LinearForwardingTable block number
Reserved		32	32	Reserved
LinearForwardingTable		512	64	Contents of LinearForwardingTable block # BlockNum contents; see Table 152 LinearForwardingTable on page 837

15.2.5.7 RANDOMFORWARDINGTABLERECORD

Table 196 RandomForwardingTableRecord

Component		Length(bits)	Offset(bits)	Description
RID	LID	16	0	LID of switch port 0
	BlockNum	16	16	RandomForwardingTable block number; see Table 155 LID/Port Block Element on page 838
Reserved		32	32	Reserved
RandomForwarding-Table		512	64	Contents of RandomForwardingTable block # BlockNum; see Table 154 RandomForwardingTable on page 838

15.2.5.8 MULTICASTFORWARDINGTABLERECORD

Table 197 MulticastForwardingTableRecord

Component		Length(bits)	Offset(bits)	Description
RID	LID	16	0	LID of switch port 0
	Position	4	16	Position field of this attribute as defined in 14.2.5.12 Multi-castForwardingTable on page 838.
	Reserved	3	20	Reserved.
	BlockNum	9	23	Pointer to a block of 32 PortMask entries for this attribute. See 14.2.5.12 MulticastForwardingTable on page 838
Reserved		32	32	Reserved
MulticastForwarding-Table		512	64	Contents of MulticastForwardingTable block identified by BlockNum and Position; see Table 156 MulticastForward-ingTable on page 839

15.2.5.9 VLARBITRATIONTABLERECORD

Table 198 VLArbitrationTableRecord

Component		Length(bits)	Offset(bits)	Description
RID	LID	16	0	For a CA or router: LID of the port. For a switch: LID of switch port 0.
	OutputPortNum	8	16	For a switch: output port number For a channel adapter or router: reserved
	BlockNum	8	24	VL/Weight block number; see Table 151 VL/Weight Block Element on page 837
Reserved		32	32	Reserved

Table 198 VLArbitrationTableRecord (Continued)

Component	Length(bits)	Offset(bits)	Description
VLArbitrationTable	512	64	Contents of VLArbitrationTable block # BlockNum; see Table 150 VLArbitrationTable on page 837

15.2.5.10 SMINFORECORD

Table 199 SMInfoRecord

Component		Length(bits)	Offset(bits)	Description
RID	LID	16	0	LID of the port
	Reserved	16	16	Reserved
SMInfo		168	32	SMInfo attribute contents; see Table 158 SMInfo on page 840

15.2.5.11 P_KEYTABLERECORD

Table 200 P_KeyTableRecord

Component		Length(bits)	Offset(bits)	Description
RID	LID	16	0	For a CA or router: LID of the port. For a switch: LID of switch port 0.
	BlockNum	16	16	P_Key Table block number
	PortNum	8	32	For a switch: port number For a CA or router: reserved
Reserved		24	40	Reserved
P_Key Table		512	64	Contents of P_KeyTable block number BlockNum; see Table 147 P_KeyTable on page 835

15.2.5.12 INFORMINFORECORD

Table 201 InformInfoRecord

Component		Length(bits)	Offset(bits)	Description
RID	SubscriberGID	128	0	GID of the subscriber port
	Enum	16	128	Identifier unique among all subscriptions requested through port
Reserved		48	144	Reserved

Table 201 InformInfoRecord (Continued)

Component	Length(bits)	Offset(bits)	Description
InformInfo	288	192	InformInfo attribute contents; see Table 119 InformInfo on page 739

15.2.5.13 LINKRECORD

Table 202 LinkRecord

Component		Length(bits)	Offset(bits)	Description
RID	FromLID	16	0	For a CA or router: LID of a port connected by a link to another port. For a switch: LID of a switch port 0 whose port number FromPort is connected by a link to another port.
	FromPort	8	16	For a CA or router: 0 For a switch: port number, other than port 0
ToPort		8	24	If the port specified by FromLID/FromPort is connected by a link to • a CA or router port: ToPort is 0 • a switch port: ToPort is the port number of that switch port
ToLID		16	32	If the port specified by FromLID/FromPort is connected by a link to • a CA or router port: ToLID is the LID of that CA or router port • a switch port: ToLID is the LID of port 0 of that switch

LinkRecords are synthesized by SA to serve as informational topology data for management entities in need of such data. It is intended to identify physical links between specific port pairs. The “From” side of the LinkRecord refers to where the link departs and the “To” side of the LinkRecords refers to where the link arrives. Note that since links are bi-directional, two LinkRecords will be synthesized for each link in a subnet.

15.2.5.14 SERVICERECORD

Table 203 ServiceRecord

Component		Length(bits)	Offset(bits)	Description
RID	ServiceID	64	0	The identifier of the service on the port specified by ServiceGID
	ServiceGID	128	64	The port GID for the service.
	ServiceP_Key	16	192	The P_Key used in contacting the service.
Reserved		16	208	Reserved for alignment.
ServiceLease		32	224	Lease period remaining for this service, in seconds. 0xFFFFFFFF is an indefinite lease.

Table 203 ServiceRecord (Continued)

Component		Length(bits)	Offset(bits)	Description
ServiceKey		128	256	Key value that may be associated with the Service-Name; see discussion following.
ServiceName		512	384	UTF-8 encoded, null-terminated character string used to identify the service (e.g., "BIS.IBTA").
Note: more than one	ServiceData8.1 … ServiceData8.16	8×16=128	896	Data for this ServiceRecord; content is opaque to SA. One ComponentMask bit per byte. See discussion.
	ServiceData16.1 … ServiceData16.8	16×8=128	1024	Data for this ServiceRecord; content is opaque to SA. One ComponentMask bit per 16 bits. See discussion.
	ServiceData32.1 … ServiceData32.4	32×4=128	1152	Data for this ServiceRecord; content is opaque to SA. One ComponentMask bit per 32 bits. See discussion.
	ServiceData64.1 … ServiceData64.2	64×2=128	1280	Data for this ServiceRecord; content is opaque to SA. One ComponentMask bit per 64 bits. See discussion.

ServiceRecords provide a first level or "bootstrap" advertisement of basic services that cannot be found prior to a query of SA. These could be services such as a boot service or a name service. Managers which define events (trap numbers) should create ServiceRecords so entities wishing to subscribe to their events can find them. ServiceRecords are not intended to substitute for a full function, highly scalable, network directory service.

15.2.5.14.1 SERVICEP_KEY

The ServiceP_Key is a P_Key which may be used to contact the service. As required by 15.4.1.2 Access Restrictions For Other Attributes on page 922, SA returns to a requesting port only ServiceRecords whose ServiceP_Key matches an entry in that port's P_Key Table.

C15-0.1.13: SA **shall** reject as invalid any attempt to create, modify, or delete a ServiceRecord in which the ServiceP_Key is not present in the P_Key Tables of both the port identified by the ServiceGID and the port from which the request came.

15.2.5.14.2 SERVICEKEY

The ServiceKey may be used to allow authentication of the creation, replacement and deletion of ServiceRecords with selected ServiceNames.

C15-0.1.14: SA **shall** provide a means for associating ServiceNames with ServiceKeys.

This association is vendor-specific. Implementations may differ in such aspects as how the association is created and maintained; how many different associations may be held by SA; whether a ServiceName may be

associated with multiple ServiceKeys; whether different ServiceNames may have the same ServiceKey; etc. The set of {ServiceName, ServiceKey} associations can be retrieved, but not SubnAdmSet(), by using the ServiceAssociationRecord; see 15.2.5.15 ServiceAssociationRecord on page 899.

C15-0.1.15: Requests to SubnAdmSet(ServiceRecord) or SubnAdmDelete(ServiceRecord) **shall** be performed as if the procedure described below were used. In what follows, Ns is the ServiceName in the attribute supplied with the MAD, Ks is the ServiceKey, and Rs is the RID. In all cases it is assumed that the ComponentMask is appropriately set.

1) If no {ServiceName, ServiceKey} association exists with ServiceName=Ns, ignore Ks and perform the requested operation, i.e.:
   a) If the method is SubnAdmDelete(), then
      i) if there is a stored ServiceRecord with a RID matching Rs, then delete that stored ServiceRecord.
      ii) otherwise reject the operation.
   b) If the method is SubnAdmSet(), then
      i) if there is a stored ServiceRecord with a RID matching Rs, replace that stored ServiceRecord with the one supplied.
      ii) otherwise add the supplied ServiceRecord.
2) Otherwise, if Ks does not equal any ServiceKey associated with Ns, i.e., no {Ns, Ks} association exists, then reject the operation.

- Note, having passed step 2 means that in steps 3 and 4, an {Ns, Ks} association is known to exist.

3) If the method is SubnAdmSet(), then:
   a) if there is no stored ServiceRecord with a RID matching Rs, add the supplied ServiceRecord.
   b) if there is a stored ServiceRecord with a RID matching Rs and a ServiceKey matching Ks, then replace the stored ServiceRecord with the one supplied.
   c) otherwise, reject the operation
4) If the method is SubnAdmDelete(), then:
   a) if there is a stored ServiceRecord with a RID matching Rs and a ServiceKey matching Ks, then delete that ServiceRecord.
   b) otherwise, reject the operation.

An implication of C15-0.1.15: is that requests to add, delete, or replace a ServiceRecord, whether successful or not, have no effect on associations. SubnAdmSet()s and SubnAdmDelete()s do not change which ServiceNames are associated with ServiceKeys or whether specific Service-

Names are associated with specific ServiceKeys. In particular, adding a ServiceRecord with a previously unassociated ServiceName does *not* create a new association. When there is no association with a ServiceName, the ServiceKey is always ignored.

Queries of ServiceRecords, e.g., using SubnAdmGet() or SubnAdmGetTable() methods, match the ServiceKey like any other component. It is anticipated that most queries will use the ComponentMask to wildcard ServiceKey.

As specified in 15.4.1.2 Access Restrictions For Other Attributes on page 922, SA returns ServiceRecords with the ServiceKey replaced by 0 except when responding to a trusted subnet manager.

15.2.5.14.3 SERVICELEASE

ServiceLease values that are long or indefinite can conceivably span a time period that includes subnet reinitialization, including reinitialization of SA. Whether such ServiceRecords are nonvolatile, i.e., whether or not reinitialized SA begins with those ServiceRecords in place, is vendor-dependent.

SA may impose a maximum ServiceLease. That is, it may reject attempts to write ServiceRecords with a ServiceLease period that is indefinite or larger than some amount. Whether this is done, and the way the maximum is determined, is vendor-dependent.

C15-0.1.16: SA **shall** begin counting down each non-indefinite ServiceLease period immediately on creation or modification of a ServiceRecord. The ServiceLease value returned in a ServiceRecord query **shall** be the time remaining in its ServiceLease.

C15-0.1.17: When a ServiceRecord's ServiceLease period expires, SA **shall** delete the ServiceRecord.

15.2.5.14.4 SERVICEDATA

The ServiceData8.1 through ServiceData64.2 components together constitutes a 64-byte area in which any data may be placed. It is intended to be a convenient way for a service to provide its clients with some initial data.

In addition, this 64-byte area is formally divided into a total of 30 components—16 8-bit components, then 8 16-bit components, etc.—thereby assigning ComponentMask bits to variously-sized segments of the data. This allows query operations to be used which match parts of the ServiceData, making it possible, for example, for service-specific parts of the ServiceData to serve as a binary-coded extension to the ServiceName for purposes of lookup.

Since the breakup of ServiceData into components is purely formal (no semantics apply to it); and since the matching performed in a query is a simple bitwise match; it's the case that adjacent formal components of ServiceData can in effect be arbitrarily concatenated into larger entities if convenient. For example, a 32-bit binary value to be used in matching could occupy the first four bytes of ServiceData (components ServiceData8.1 through ServiceData8.4). It can be matched against by setting to 1 the first four bits of ComponentMask corresponding to those first four formal ServiceData components. The client doing the lookup must, of course, be cognizant of the way the service it is requesting uses these components.

15.2.5.15 SERVICEASSOCIATIONRECORD

Table 204 ServiceAssociationRecord

Component	Length(bits)	Offset(bits)	Description
ServiceKey	128	0	Key value that may be associated with the Service-Name; see discussion following.
ServiceName	512	128	UTF-8 encoded, null-terminated character string used to identify the service (e.g., "BIS.IBTA").

This attribute cannot be modified using SubnAdmSet(). Its purpose is to allow SA of another vendor to obtain all the {ServiceName, ServiceKey} associations currently stored in SA by using a SubnAdmGetTable(ServiceAssociationRecord) with a ComponentMask of all zeros.

15.2.5.16 PATHRECORD

Table 205 PathRecord

Component	Length(bits)	Offset(bits)	Required For GetTable Request	Description
Reserved	32	0		Reserved
Reserved	32	32		Offset for alignment
DGID	128	64		Destination GID to establish path to
SGID	128	192	X	Source GID to establish path from; **shall** be a unicast GID
DLID	16	320		Destination LID
SLID	16	336		Source LID; **shall** be a unicast LID

Table 205 PathRecord (Continued)

Component	Length(bits)	Offset(bits)	Required For GetTable Request	Description
RawTraffic	1	352		Raw Packet path 0 - IB Packet (P_Key must be valid) 1 - Raw Packet traffic (No P_Key)
Reserved	3	353		Reserved
FlowLabel	20	356		Flow Label
HopLimit	8	376		Hop limit
TClass	8	384		Traffic Class
Reversible	1	392		In a query request: • 1 indicates that a reversible path is required; see 13.5.4 Response Generation and Reversible Paths on page 768 • 0 indicates that reversibility is not required In a query response: • 1 indicates the path is reversible • 0 indicates the path is not reversible
NumbPath	7	393	X	In a query request: Maximum number of paths to return for each unique SGID-DGID combination. If more paths that satisfy the PathRecord query exist for a given SGID-DGID combination, only NumbPath paths **shall** be returned (implementation defined). In a query response: undefined.
P_Key	16	400		Partition Key for this path
Reserved	12	416		Reserved
SL	4	428		Service level
MtuSelector	2	432		In a query request: 3-largest MTU available If MTU is specified (i.e., the ComponentMask bit for MTU is 1): 0-greater than MTU specified 1-less than MTU specified 2-exactly the MTU specified Otherwise, this component is ignored in a query request. In a query response: value **shall** be 2 indicating path has exactly the MTU specified.

Table 205 PathRecord (Continued)

Component	Length(bits)	Offset(bits)	Required For GetTable Request	Description
Mtu	6	434		Enumeration of the MTU required: 1: 256 2: 512 3: 1024 4: 2048 5: 4096 0,6-63: reserved In a query request, if MtuSelector is not specified (i.e., the ComponentMask bit for MtuSelector is 0) or if it is specified and has a value of 3, this component is ignored.
RateSelector	2	440		In a query request: 3-largest rate available If Rate is specified (i.e., the ComponentMask bit for Rate is 1): 0-greater than Rate specified 1-less than Rate specified 2-exactly the Rate specified Otherwise, this component is ignored in a query request. In a query response: value **shall** be 2 indicating path has exactly the Rate specified.
Rate	6	442		Enumeration of the rate: 2: 2.5 Gb/sec. 3: 10 Gb/sec. 4: 30 Gb/sec. 5: 5 Gb/sec. 6: 20 Gb/sec. 7: 40 Gb/sec. 8: 60 Gb/sec. 9: 80 Gb/sec. 10: 120 Gb/sec. 0, 1, 11-63: reserved In a query request, if RateSelector is not specified (i.e., the ComponentMaskBit for RateSelector is 0) or if it is specified and has a value of 3, this component is ignored. SA shall return to an endport only PathRecord:Rate values that the endport can support, as indicated by the values provided in that endport's PortInfo components LinkSpeedSupported and LnkWidthSupported, unless PortInfo:IsOptionalIPDSupported=1; in the latter case, any rate value can be returned.

Table 205 PathRecord (Continued)

Component	Length(bits)	Offset(bits)	Required For GetTable Request	Description
PacketLife-TimeSelector	2	448		In a query request: 3-smallest PacketLifeTime available If PacketLifeTime is specified (i.e., the Component-Mask bit for PacketLifeTime is 1): 0-greater than PacketLifeTime specified 1-less than PacketLifeTime specified 2-exactly the PacketLifeTime specified Otherwise, this component is ignored in a query request. In a query response: value **shall** be 2 indicating path has exactly the PacketLifeTime specified.
PacketLife-Time	6	450		Maximum time for a packet to traverse this path, where time in microseconds is derived from time = 4.096 µsec * $2^{PacketLifeTime}$. In a query request, if PacketLifeTimeSelector is not specified (i.e., the ComponentMask bit for PacketLife-TimeSelector is 0) or if it is specified and has a value of 3, this component is ignored. For loopback paths, **shall** be zero.
Preference	8	456		In responses containing multiple paths: Indicates an order of preference among paths returned that over-rides differences that may be indicated by MTU, Rate, and/or PacketLifetime. Larger numbers indicate worse paths. Identical numbers indicate no prefer-ence among paths aside from that which may be clear from other PathRecord fields. The precise meaning of this field and the features on which it is based is implementation dependent.
Reserved	48	464		Reserved

The PathRecord is used to request routing information between endn-odes. Its results are required to create connections and perform other tasks. The data returned in a PathRecord is usually generated, based on the routing algorithm used by the SM.

For unicast communication, a PathRecord specifying a path from SGID A to DGID B exists if all the following conditions hold:

- there is a LID-routed path from port A to port B,
- PortInfo:PortState of port A is Active,
- PortInfo:PortState is Armed or Active for all intermediate switch and router ports in the path from port A to port B, and

- PortInfo:PortState of port B is Armed or Active.

For multicast communication, a PathRecord specifying a path from SGID A to MGID (multicast GID) B exists if PortInfo:PortState of port A is Active, and port A belongs to the multicast group with MGID B.

Requesters of PathRecords can use the Administration Query Subsystem (15.4.4 Administration Query Subsystem on page 923) to request paths with desired properties. Using the ComponentMask, the requester can build a SubnAdmGetTable() request and supply the known fields in the attribute; the reply from SA will supply the response entries which match the request. For example, by setting the ComponentMask used to cause everything but the SGID and DGID to be ignored, a SubnAdmGetTable() will return PathRecords for all paths from the SLID to the DLID. By selectively specifying the qualities desired, a path with any given qualities can be requested.

In calculating PMTU, SL, and rate for a path terminating at a base switch port 0, the PortInfo components for that base switch port 0 are ignored.

It is legal to request a PathRecord from a port to itself. Such “loopback” PathRecords are useful for diagnostic purposes as well as for normal traffic between clients on the same endnode. SA handles such queries as though the port were connected by a point-to-point link to another port having identical capabilities. This implies that certain characteristics, for example NeighborMTU, may be implicitly changed; this is feasible because loopback traffic never reaches the physical link.

Normally the DGID is known (or at some point learned, as with, e.g., name service). But during a “boot” sequence, it may be useful to leave it unspecified, thus returning paths to all endnodes reachable from an SGID.

The equivalent of an operating system's “bus walk” operation, used to find out which device slots are populated, can be accomplished using a single PathRecord query. For example, consider a SubnAdmGetTable(PathRecord) query with:

- SGID set to a GID of some port
- NumbPath set to 1
- ComponentMask all 0 except for SGID and NumbPath positions, which are 1.

Then SA will return a table containing paths from the physical port identified by the SGID to every physical port reachable from that SGID port, given the partitioning in force when the query is done (see 15.4.1 Restrictions on Access on page 921). Multiple paths to each physical destination port may be returned: Each physical destination port may have multiple

GUIDs if its PortInfo:GUIDCap is greater than 1, and each of those defines a separate GID.

Since NumbPath=1, and is defined as the number of paths connecting each SGID and DGID, there will be only one path returned to each reachable DGID even if both the SGID port and the DGID port have multiple LIDs (i.e., their LMC values are not 0).

ComponentMask matching for PathRecord queries involves several exceptions to the matching mechanism specified in C15-0.1.28:. Refer to the descriptions in Table 205 PathRecord on page 899 for details on using NumbPath, MtuSelector/Mtu, RateSelector/Rate, and PacketLifeTimeSelector/PacketLifeTime in query requests.

The following example shows a PathRecord query request, seeking as many as two paths with a specified set of characteristics. Table 206 Example PathRecord Request MAD Header Fields on page 904 shows the header fields of this request; only those header fields whose values differ from the defaults are shown.

Table 206 Example PathRecord Request MAD Header Fields

Component	Value	Description
MADHeader:MgmtClass	0x03	Specifying SubnAdm class
MADHeader:Method	0x12	Specifying SubnAdmGetTable()
MADHeader:TransactionID	0x0000000011223344	Transaction ID (to be returned in the response)
MADHeader:AttributeID	0x0035	Specifying PathRecord
RMPPHeader:RMPP-Flags.Active	0	Information flags:RMPP is not active
SAHeader:SM_Key	0	SM_Key value is zero; see 15.4.1 Restrictions on Access on page 921
SAHeader:ComponentMask	0x00000000000F904C	Selecting those components to be used; see ComponentMask Bit column in Table 207 on page 905:

Table 207 Example PathRecord Request Data on page 905 shows what the data fields in the request would look like if up to two paths are requested to a particular port The GID of that port is assumed already available; it might, for example, be the port on which some service is located, obtained by doing a lookup of a ServiceRecord. The specifications for the paths being requested are:

- the MTU is no larger than 1024,
- the rate must be at 2.5 Gb/sec.,
- the path must be in the partition to which the requester has access,

- using service level 8,
- for any PacketLifeTime cost,
- for any TClass,
- for IB traffic,
- any flow label,
- or any number of “hops”

Table 207 Example PathRecord Request Data

Component	Component Mask Bit	Value	Meaning/Implication
Reserved	0	0	
Reserved	0	0	
DGID	1	GID	GID of the port to which paths are requested
SGID	1	Requester GID	GID of the port from which requested paths are to begin
DLID	0	0	Indicates to SA that the path may be to any port on the destination node.
SLID	0	0	Indicates to SA that the path may be from any port on the local node.
RawTraffic	1	0	IB traffic (P_Key will be valid)
Reserved	0	0	ignored - set to 0
FlowLabel	0	0	Flow Label
HopLimit	0	0	Hop Limit
TClass	0	0	Traffic Class
Reversible	0	0	Non-reversible path is acceptable
NumbPath	1	2	Requesting only 2 paths which meet these specifications
P_Key	0	0	No P_Key specified for this path
Reserved	0	0	
SL	1	0x8	Desired SL for this path (SL 8)
MtuSelector	1	1	Paths must have an MTU less than 2048
Mtu	1	4	Mtu is less than 2048, i.e., 1024 or lower
RateSelector	1	2	Path rate must be exactly 2.5Gb/sec.
Rate	1	2	Rate is equal to 2.5Gb/sec.
PacketLife-TimeSelector	0	0	Return paths with any PacketLifeTime

Table 207 Example PathRecord Request Data

Component	Component Mask Bit	Value	Meaning/Implication
PacketLife-Time	0	0	
Preference	0	0	Not used on requests
Reserved	0	0	Padding

For this example the following is a possible resulting response header and the PathRecords found in the data field of the response; again, only the fields whose values differ from defaults are shown:

Table 208 Example PathRecord Response MAD Header Fields

Component	Value	Meaning/Implication
MADHeader:MgmtClass	0x03	Specifying SubnAdm class
MADHeader:Method	0x92	SubnAdmGetTableResp()
MADHeader:Status	0	Good Status
MADHeader:TransactionID	0x0000000011223344	Same as in request
MADHeader:AttributeID	0x0035	Specifying PathRecord
RMPPHeader:RMPPType	1	Specifying a data packet
RMPPHeader:RRespTime	0x1F	No time value provided
RMPPHeader:RMPPFlags.First	1	Information flags: first packet
RMPPHeader:RMPPFlags.Last	1	Information flags: last packet
RMPPHeader:RMPPFlags.Active	1	Information flags:RMPP is active
RMPPHeader:RMPPStatus	0	Normal data packet status
RMPPHeader:SegmentNumber	00000001	First segment
RMPPHeader:PayLoadLength	148	2*8*AttributeOffset + sizeof (SAHheader) in bytes
SAHeader:SM_Key	0x0	SM_Key value from query
SAHeader:AttributeOffset	0x08	Offset from an attribute to the next in the data, in 8-byte words
SAHeader:ComponentMask	0x00000000000F904C	ComponentMask from query

The following are the attributes in the data field of the resulting response:

Table 209 Example PathRecord Response Data

Component	Value	Meaning/Implication
Reserved	0	
Reserved	0	
DGID	GID	GID of the port at which this path ends
SGID	Requester GID	GID of the port from which this path is to begin
DLID	0x0008	The LID assigned to the port where this path ends
SLID	0x000A	The LID assigned to the port where this path begins
RawTraffic	0	IB traffic (P_Key will be valid)
Reserved	0	ignored - set to 0
FlowLabel	0	Flow Label
HopLimit	0	Default Hop Limit since this is an intra-subnet DGID
TClass	0	Traffic Class
Reversible	0	Path is not reversible
NumbPath	0	Ignored - set to 0
P_Key	0x1234	P_Key in use for this path
Reserved	0	Ignored - set to 0
SL	0x8	SL of this path is 8
MtuSelector	2	Path Mtu is exact
Mtu	3	Specified Mtu is exactly 1024
RateSelector	2	Path rate is exact
Rate	2	Path rate is exactly 2.5Gb
PacketLifeTimeSelector	2	Path PacketLifeTime is exact.
PacketLifeTime	2	Path PacketLifeTime is 4.096µsec * 2^2 = 16.384µsec exactly.
Preference	0	0
Reserved	0	
Reserved	0	
Reserved	0	
DGID	GID	GID of the port at which this path ends
SGID	Requester GID	GID of the port from which this path is to begin
DLID	0x0009	The LID assigned to the port where this path ends

Table 209 Example PathRecord Response Data (Continued)

Component	Value	Meaning/Implication
SLID	0x000A	The LID assigned to the port where this path begins
RawTraffic	0	IB traffic (P_Key will be valid)
Reserved	0	Ignored - set to 0
FlowLabel	0	FlowLabel
HopLimit	0	Default HopLimit since this is an intra-subnet DGID
TClass	0	Traffic Class
Reversible	0	Path is not reversible
NumbPath	0	Ignored - set to 0
P_Key	0x1234	P_Key in use for this path
Reserved	0	Ignored - set to 0
SL	0x8	SL of this path is 8
MtuSelector	2	Path Mtu is exact
Mtu	2	Path Mtu is exactly 512
RateSelector	2	Path rate is exact
Rate	2	Path rate is exactly 2.5Gb/sec.
PacketLifeTimeSelector	2	Path PacketLifeTime is exact.
PacketLifeTime	0x0A	Path PacketLifeTime is 4.096µsec * 2^{10} = 4.2msec exactly.
Preference	0	0
Reserved	0	Reserved

15.2.5.17 MCMEMBERRECORD

Table 210 MCMemberRecord

Component		Length (bits)	Offset (bits)	Description
RID	MGID	128	0	Multicast GID address for this multicast group. See discussion below
	PortGID	128	128	Valid GID of the endport joining this multicast group.
Q_Key		32	256	Q_Key to be used by this multicast group.
MLID		16	288	Multicast LID for this multicast group, assigned by SA at creation time.
MTUSelector		2	304	Semantics identical to PathRecord component MtuSelector on page 900
MTU		6	306	Semantics identical to PathRecord component Mtu on page 901

Table 210 MCMemberRecord (Continued)

Component	Length (bits)	Offset (bits)	Description
TClass	8	312	Semantics identical to PathRecord component TClass on page 900
P_Key	16	320	Partition key for this multicast group. This partition key shall indicate full membership (see 10.9.1.1 Limited and Full Membership on page 524). All members of the multicast group shall have full membership in the partition indicated by this partition key.
RateSelector	2	336	Semantics identical to PathRecord component RateSelector on page 901
Rate	6	338	Semantics identical to PathRecord component Rate on page 901
PacketLifeTimeSelector	2	344	Semantics identical to PathRecord component PacketLifeTimeSelector on page 902
PacketLifeTime	6	346	Maximum estimated time for a packet to traverse a path within the multicast group.
SL	4	352	Semantics identical to PathRecord component SL on page 900
FlowLabel	20	356	Semantics identical to PathRecord component FlowLabel on page 900
HopLimit	8	376	Semantics identical to PathRecord component HopLimit on page 900
Scope	4	384	Semantics identical to MGID Address Scope Table 3 Multicast Address Scope on page 146. See discussion below
JoinState	4	388	Join/Leave Status requested by the port. See discussion below. bit 0: FullMember: Include/delete this endport from the multicast group as a member sender and receiver. bit 1: NonMember: Include/delete this endport from the multicast group as a non-member sender and receiver. bit 2: SendOnlyNonMember: Include/delete this endport from the multicast group as a non-member sender only. bit 3: Reserved
ProxyJoin	1	392	Proxy join: This is computed by the SA. It is ignored on SubnAdmSet(), and always returned as zero except for the case of a trusted request (see 15.4.1.2 Access Restrictions For Other Attributes on page 922). • 0: Join was performed by the endport identified by PortGID. • 1: Join was performed on behalf of the endport identified by PortGID by another port within the same partition.
Reserved	23	393	Reserved

An entity that wishes to create, join or leave an IBA multicast group, can do so using the SubnAdmSet() and SubnAdmDelete() methods with the MCMemberRecord attribute. This section only defines the interface to SA and SM that provides them with the information needed to set up routing of multicast packets. In order for an endnode to receive multicast packets, other additional operations need to be performed at that endnode; see 10.5 Multicast Services on page 465.

Note: This version of the specification does not provide management support for raw multicast.

SubnAdmSet() method is used to create or join a multicast group. SubnAdmDelete() method is used to delete or leave a multicast group.

o15-0.1.1: If SA supports UD multicast, then a SubnAdmSet(MCMemberRecord) which would result in the port being added to the multicast group implies that the SM **shall** program routers and switches with the new multicast information. If the SM is unable to program the fabric with the new multicast information, SA **shall** return an error status of ERR_NO_RESOURCES in its response to the corresponding SubnAdmSet() method.

The ProxyJoin component is used by SA to keep track of join requests for multicast groups by ports other than the port specified by PortGID, and to ensure that only ports within the partition can delete a MCMemberRecord for the port specified by PortGID, and then only if it was a proxy join. In the case of a non-proxy join, SA allows only the port specified by PortGID to request the deletion.

o15-0.1.2: IThis compliance statement is obsolete and has been replaced by statement o15-0.2.1:.

o15-0.2.1: If SA supports UD multicast, then if SA receives a SubnAdmSet() or SubnAdmDelete() method that would modify an existing MCMemberRecord, SA **shall not** modify that MCMemberRecord and **shall** return an error status of ERR_REQ_INVALID in response in the following cases:

- Saved MCMemberRecord.ProxyJoin is not set and the request is issued by a requester with a GID other than the PortGID.
- Saved MCMemberRecord.ProxyJoin is set and the requester is no part of the partition for that MCMemberRecord.

o15-0.1.3: If SA supports UD multicast, then if SA receives a SubnAdmSet() or SubnAdmDelete() method without the required components as specified below, SA **shall** return an error status of ERR_REQ_INSUFFICIENT_COMPONENTS in its response. Exactly which components are required depends upon the type of request.

In 15.2.5.17.2 Creating a Multicast Group on page 912 and 15.2.5.17.3 Joining a Multicast Group on page 913 below, the term “unrealizable” is used. “Unrealizable” means that a requested multicast group creation or join is impossible because the request specifies properties that are physically impossible to achieve given the hardware configuration of the subnet. For example, creation of a group with an MTU or Rate larger than

that supported anywhere in a subnet is unrealizable, as is a group with a PacketLifeTime shorter than that required to deliver packets between any of the endnodes of the subnet. Similarly, a join request is unrealizable if, for example, there is no path to the port joining which is able to support the MTU, Rate, etc., of the multicast group.

15.2.5.17.1 GROUP MEMBERSHIP

An endport must specify the type of multicast subscription or deletion that it wants. The MCMemberRecord:JoinState component indicates the membership qualities a port wishes to add (in joining or creating a group) or remove (in leaving a group). The meanings of the MCMember-Record:JoinState bits are:

- FullMember: Group messages are routed both to and from the port. The port is considered a member for purposes of group creation and deletion, i.e.: if no member ports with FullMember=1 remain, the group may be deleted; otherwise it may not.
- NonMember: Group messages are routed both to and from the port. The port is not considered a member for purposes of group creation/deletion.
- SendOnlyNonMember: Group messages are only routed from the port; none are routed to the port. The port is not considered a member for purposes of group creation/deletion.

Any combination of the MCMemberRecord:JoinState bits may be 1. When multiple bits are 1, the qualities of the port's type of membership are the union of the qualities specified by the bits that are 1. For example, if either or both of MCMemberRecord:JoinState.FullMember and MCMember-Record:JoinState.NonMember are 1, group messages are routed both to and from the port no matter what value MCMemberRecord:Join-State.SendOnlyNonMember has; and if MCMemberRecord:Join-State.FullMember is 1, the port is counted as a member for purposes of creation/deletion no matter what the settings of MCMemberRecord:Join-State.NonMember or MCMemberRecord:JoinState.SendOnlyNon-Member may be.

MCMemberRecord:JoinState.FullMember bit must be set to 1 in the SubnAdmSet() request that creates a multicast group. Subsequent join or leave requests referencing that multicast group can contain any combination of the MCMemberRecord:JoinState bits. When SA receives a join request with the same RID as an already existing MCMemberRecord, SA updates the MCMemberRecord:JoinState component of the MCMember-Record to be the logical OR of the MCMemberRecord:JoinState in the request with the current JoinState maintained for that MCMemberRecord. When SA receives a leave request, it updates the referenced MCMember-Record's MCMemberRecord:JoinState components to be the logical AND of the MCMemberRecord's current MCMemberRecord:JoinState with the

logical NOT of the MCMemberRecord:JoinState in the request. If this leave request causes there to be no MCMemberRecord for a multicast group that has the MCMemberRecord:JoinState.FullMember bit set, the multicast group may be deleted by the SM/SA.

15.2.5.17.2 CREATING A MULTICAST GROUP

o15-0.1.4: This compliance statement is obsolete and has been replaced by statement o15-0.2.2:.

o15-0.2.2: If SA supports UD multicast, then SA **shall** create a multicast group if it receives a SubnAdmSet() method for a MCMemberRecord, with the MGID set to 0 and the MCMemberRecord:JoinState.FullMember bit set to 1. The required components in the MCMemberRecord for the group to be created are P_Key, Q_Key, SL, FlowLabel, TClass, JoinState and PortGID (see o15-0.1.3:) with the corresponding bits in the Component-Mask set. All other components may be wildcarded. This results in an implicit join for the port specified by PortGID.

o15-0.1.5: If SA supports UD multicast, then if SA receives a request to create a multicast group with the MGID set to 0, the MGID that it creates **shall** be of the following format:

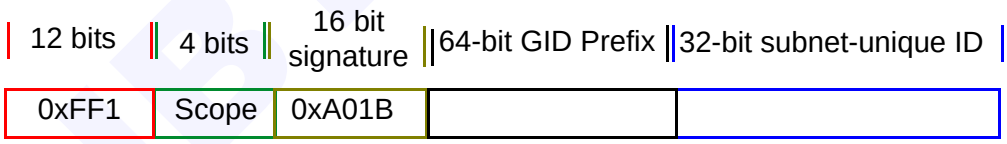


Figure 196 SA-Created Multicast GID Format

where the Scope bits are provided by the client in the MCMember-Record:Scope component and the 32-bit subnet-unique ID is chosen by SA such that MGIDs generated by SA **shall** be unique among multicast groups existing at any given time within the scope used[24]. If the multicast group creator does not provide the Scope component, SA **shall** choose the scope bits for the MGID with the broadest possible scope allowed as specified by the GID prefix.

o15-0.1.6: This compliance statement is obsolete and has been replaced by statement o15-0.2.3:.

o15-0.2.3: If SA supports UD multicast, then SA **shall** create a multicast group if it receives a SubnAdmSet() method for a MCMemberRecord with a valid MGID specified, there does not exist a multicast group with that MGID and the MCMemberRecord:JoinState.FullMember bit is set to 1. The required components in the MCMemberRecord for the group to be created **shall** be P_Key, Q_Key, SL, FlowLabel, TClass, JoinState and

24. Uniqueness across different SAs within a fabric is guaranteed by inclusion of the GID prefix

PortGID (see o15-0.1.3:) with the corresponding bits in the ComponentMask set. All other components may be wildcarded.This results in an implicit join for the port specified by PortGID. The Scope component is ignored in this case.

A multicast GID is considered to be invalid if:

- it does not comply with the rules as specified in 4.1.1 GID Usage and Properties on page 143
- it contains the SA-specific signature of “0xA01B” and has the link-local scope bits set.

Additional cases of invalidity may appear for non-local subnets and may be provided in future revisions of the specification.

The entity may also supply the other components such as HopLimit, MTU etc. during group creation time. If these components are not provided during group creation time, SA will provide them for the group. The values chosen are vendor-dependent and beyond the scope of the specification.

o15-0.1.7: If SA supports UD multicast, then a request to create a multicast group with an invalid MGID **shall** result in SA returning an error status of ERR_REQ_INVALID_GID in its response.

o15-0.1.8: If SA supports UD multicast, then if SA receives a request that would result in the creation of a multicast group with components specified that are unrealizable for its subnet, SA **shall** return an error status of ERR_REQ_INVALID in its response.

o15-0.1.9: If SA supports UD multicast, then if SA receives a request which would create a multicast group and the MCMemberRecord:JoinState.FullMember bit is not set to 1, SA shall return an error status of ERR_REQ_INVALID in its response.

15.2.5.17.3 JOINING A MULTICAST GROUP

o15-0.1.10: This statement is obsolete and has been replaced by statement o15-0.2.4:.

o15-0.2.4: If SA supports UD multicast, then SA **shall** cause an endport to join an existing multicast group if:

- it receives a SubnAdmSet() method for a MCMemberRecord, and
- the MGID is specified and matches an existing multicast group, and
- the MCMemberRecord:JoinState is not all 0s, and
- PortGID is specified and

- all other components match that existing group, either by being wildcarded or by having values identical to those specified by the component mask and in use by the group with the exception of components such as ProxyJoin and Reserved, which are ignored by SA.

o15-0.1.11: If SA supports UD multicast, then if an endport joins a multicast group as specified in o15-0.1.10:, SA **shall** replace the endport's current MCMemberRecord:JoinState component with the logical OR of the MCMemberRecord:JoinState component with the endport's current MCMemberRecord:JoinState component if the endport had joined this multicast group before.

o15-0.1.12: If SA supports UD multicast, then if an endport joins a multicast group as defined in o15-0.1.10: and o15-0.1.11: and has only the MCMemberRecord:JoinState.SendOnlyNonMember bit set to 1, then SA **shall** cause that endport to join the multicast group as a sender only.

o15-0.1.13: If SA supports UD multicast, then if SA receives a join request for a multicast group with components specified or wildcarded that are unrealizable for the port specified by PortGID, SA **shall** return an error status of ERR_REQ_INVALID in its response.

15.2.5.17.4 LEAVING & DELETING A MULTICAST GROUP

In o15-0.1.14: and o15-0.1.15: below, the term "valid MC deletion MAD" means a SubnAdmDelete(MCMemberRecord) MAD in which the ComponentMask bits are set and the values are as specified for the following components:

- the MAD's PortGID and MGID components match the PortGID and MIGID of a stored MCMemberRecord;
- and the MAD's JoinState component contains at least one bit set to 1 in the same position as that stored MCMemberRecord's JoinState has a bit set to 1, i.e., the logical AND of the two JoinState components is not all zeros;
- and the MAD's JoinState component does not have some bits set which are not set in the stored MCMemberRecord's JoinState component;
- and either
  - the stored MCMemberRecord:ProxyJoin is reset (0), and the MAD's source is the stored PortGID;
  - or the stored MCMemberRecord:ProxyJoin is set (1), (see o15-0.1.2:); and the MAD's source is a member of the partition indicated by the stored MCMemberRecord:P_Key.

o15-0.1.14: If SA supports UD multicast, then if it receives a valid MC deletion MAD, SA **shall** compute the logical AND of the matching stored

MCMemberRecord:JoinState with the logical NOT of the MAD's JoinState, and then:

- if the result is all 0s, SA **shall** cause the port to leave the multicast group. The port specified by the MAD's PortGID **shall** be removed from the multicast group specified by MGID or from all the multicast groups of which it is a member if the MGID is wildcarded;
- if the result is not all 0s, SA shall update the stored MCMemberRecord:JoinState to contain that result

If this port was the last receiving member, SA may delete the multicast group and release all resources associated with it, including resources that may exist in the fabric itself. The exact time at which this operation is done is vendor-specific. When the last receiving member leaves the group, SA must forward Trap 67 (see 14.4.10 Multicast Group Create/Delete Traps on page 880) to subscribing endnodes.

o15-0.1.15: If SA supports UD multicast, then if it receives a SubnAdmDelete(MCMemberRecord) which is not a valid MC deletion MAD, then SA **shall** return an error status of ERR_REQ_INVALID in its response and **shall** take no other action.

15.2.5.17.5 QUERYING A MULTICAST GROUP

SA can be queried for multicast groups by sending a SubnAdmGet() or a SubnAdmGetTable() request to it using the SA query mechanism (see 15.4.4 Administration Query Subsystem on page 923). SA will return one MCMemberRecord per multicast group matching the query, except in cases where trust is specified as indicated in 15.4.1.2 Access Restrictions For Other Attributes on page 922; in that case all the MCMemberRecords associated with the multicast group are returned. The MCMemberRecord will be returned with the PortGID, ProxyJoin, and the JoinState components set to 0, except where trust is specified as indicated above, in that case the actual contents for the above components will be provided.

o15-0.1.16: This compliance statement is obsolete and has been replaced by statement o15-0.2.5:.

o15-0.2.5: If SA supports UD multicast, then if it receives a SubnAdmGetTable(MCMemberRecord) with the MCMemberRecord:MGID wildcarded, then SA **shall** return a single MCMemberRecord for each multicast group that matches the query operation.

15.2.5.18 GUIDINFORECORD

Table 211 GuidInfoRecord

Component		Length(bits)	Offset(bits)	Description
RID	LID	16	0	LID of the Port
	BlockNum	8	16	GUIDInfo block number; see Table 144 GUID Block Element on page 821
	Reserved	8	24	Reserved
Reserved		32	32	Reserved
GUIDInfo		512	64	GUIDInfo attribute contents; see Table 143 GUIDInfo on page 821

15.2.5.19 TRACERECORD

Table 212 TraceRecord

Component	Length (bits)	Offset (bits)	Description
GIDPrefix	64	0	GID Prefix of the subnet within which the IDComponents are consistent. See discussion below.
IDGeneration	16	64	Generation of the IDs used in this attribute; see discussion below.
Reserved	8	80	Reserved
NodeType	8	88	Values and interpretation identical to NodeInfo:NodeType (seeTable 141 NodeInfo on page 818).
NodeID	64	96	ID of the Node
ChassisID	64	160	ID of the chassis or power domain, or 0 if ChassisGUID or equivalent information is unavailable.
EntryPortID	64	224	If NodeType is channel adapter: ID of the port used. If NodeType is router: ID of Entry Port. Otherwise: Reserved.
ExitPortID	64	288	If NodeType is router, and path continues past this router: ID of the exit port of the router. Otherwise: Reserved
EntryPort	8	352	If NodeType is switch: number of the entry port. Otherwise: Reserved.
ExitPort	8	360	If NodeType is switch, and path continues past this switch: number of the exit port. Otherwise: Reserved.

The TraceRecord attribute is used to describe the route a path takes from, to, or through a single node. The entire path is described by a table of TraceRecords returned as a response to the SubnAdmGetTraceTable() method.For purposes of this discussion, let *IDComponent* be any of the TraceRecord components NodeID, ChassisID, EntryPortID, or ExitPortID.

o15-0.1.17: This compliance statement is obsolete and has been replaced with o15-0.2.6:

o15-0.2.6: If ClassPortInfo:CapabilityMask.IsMultiPathSupported is 1 for SA, then IDComponents of a TraceRecord **shall not** be GUIDs of the entities they identify. IDComponents **shall** be comparable, i.e.: Given any two TraceRecords obtained from the same SA (as indicated by their having the same TraceRecord:GIDPrefix) with identical TraceRecord:IDGeneration values; and given corresponding IDComponents in each of the TraceRecords (both are ChassisIDs, or both NodeIDs, etc.); then the values of the two IDComponents **shall** be identical if they identify the same entity, and **shall** be different if they identify different entities.

Any identification scheme satisfying o15-0.2.6: may be used. For example, a separately maintained identifier set could be used, a formal encryption function could be applied to GUIDs, an informal method could be used to obfuscate GUIDs such as XORing them with a word of alternating 0's and 1's, etc. How well the IDComponent values used mask the identities of the entities they represent is vendor-specific.

It is the responsibility of SA from which a path trace is requested to coordinate with SAs of other subnets, if necessary, to provide useful, comparable TraceRecords to the requester. Note that while it may be convenient in some implementations, it is not necessary for all of the TraceRecords in a path trace to come from the same SA nor to have identical IDGeneration values.

The IDGeneration field may be changed by SA when modifications must be made to the mapping of entities to identifiers. As a result, programs using this facility may need to re-request path traces using SubnAdmGetTraceTable() to obtain TraceRecords whose IDComponents can be compared.

15.2.5.20 MULTIPATHRECORD

Table 213 MultiPathRecord

Component	Length (bits)	Offset (bits)	Description
RawTraffic	1	0	Semantics identical to PathRecord component RawTraffic on page 900

Table 213 MultiPathRecord (Continued)

Component	Length (bits)	Offset (bits)	Description
Reserved	3	1	Reserved
FlowLabel	20	4	Semantics identical to PathRecord component FlowLabel on page 900
HopLimit	8	24	Semantics identical to PathRecord component HopLimit on page 900
TClass	8	32	Semantics identical to PathRecord component TClass on page 900
Reversible	1	40	Semantics identical to PathRecord component Reversible on page 900
NumbPath	7	41	Maximum number of paths to return between SGIDs and DGIDs. If more paths that satisfy the query exist between the SGIDs and the DGIDs, only NumbPath paths **shall** be returned (implementation defined).
P_Key	16	48	Semantics identical to PathRecord component P_Key on page 900
Reserved	12	64	Reserved
SL	4	76	Semantics identical to PathRecord component SL on page 900
MtuSelector	2	80	Semantics identical to PathRecord component MtuSelector on page 900
Mtu	6	82	Semantics identical to PathRecord component Mtu on page 901
RateSelector	2	88	Semantics identical to PathRecord component RateSelector on page 901
Rate	6	90	Semantics identical to PathRecord component Rate on page 901
PacketLife-TimeSelector	2	96	Semantics identical to PathRecord component PacketLifeTimeSelector on page 902
PacketLife-Time	6	98	Semantics identical to PathRecord component PacketLifeTime on page 902
Reserved	8	104	Reserved.
Independence-Selector	2	112	1 - paths that are as fault-independent as possible 0, 2, 3 - reserved
Reserved	6	114	Reserved
SGIDCount	8	120	Number of SGIDs at the start of the SDGIDn array below. **Shall** be at least 1.
DGIDCount	8	128	Number of DGIDs following SGIDs in the SDGIDn array below. **Shall** be at least 1.
Reserved	56	136	Offset for alignment
SDGID1	128	192	Source GID 1; **shall** be a unicast GID.
SDGID2	128	320	Source or Destination GID 2; **shall** be a unicast GID.

Table 213 MultiPathRecord (Continued)

Component	Length (bits)	Offset (bits)	Description
...	128	...	Source or Destination GIDs; **shall** be unicast GIDs.
SDGIDn	128	192 + 128×(n-1)	Last Destination GID; **shall** be a unicast GID. The total number of SDGIDs **shall** equal SGIDCount + DGIDCount.

The MultiPathRecord attribute can be used to request multiple paths between a set of sources and a set of destinations, for example to find independent paths suitable for increasing availability and/or performance. It may also be used to request a single path between any of a set of sources and any of a set of destinations. There is no requirement that the number of sources be equal to the number of destinations.

o15-0.1.18: This statement is obsolete and has been replaced by statement o15-0.2.7:.

o15-0.2.7: If ClassPortInfo:CapabilityMask.IsMultiPathSupported is 1 for SA, then SA **shall** respond to a SubnAdmGetMulti() containing a valid MultiPathRecord attribute with a set of zero or more PathRecords satisfying the constraints indicated in the MultiPathRecord received. The PathRecord Attribute ID **shall** be used in the response.

Since the length of the list of SDGID elements in MultiPathRecord may vary from use to use, the length of the MultiPathRecord attribute is variable. Its size may exceed the maximum size that will fit in an SA MAD (192 bytes). Its minimum and maximum lengths are 56 bytes and 8184 bytes, respectively, corresponding to the minimum and maximum values of the SGIDCount and DGIDCount fields (respectively 1 and 255). The SubnAdmGetMulti() method allows this variability by using the reliable multi-packet protocol 15.3 Reliable Multi-Packet Transaction Protocol on page 919 to both send an attribute and receive the response.

The purpose of the IndependenceSelector component is to allow requesting paths that are fault-independent of each other, and therefore are, for example, appropriate for use as alternate paths in Automatic Path Migration (see 17.2.8 Automatic Path Migration on page 1031).

If fewer paths are requested than exist between the sources and the destinations, it is recommended that among the paths that satisfy the other specified qualities, those paths with better preferences be returned.

15.3 RELIABLE MULTI-PACKET TRANSACTION PROTOCOL

SA uses the Reliable Multi-Packet Transaction Protocol (RMPP) defined in 13.6 Reliable Multi-Packet Transaction Protocol on page 770 in conjunction with the methods SubnAdmGetTable(), SubnAdmGetTable-

Resp(), SubnAdmGetTraceTable(), SubnAdmGetMulti(), and SubnAdmGetMultiResp().

The AttributeOffset field of the SA header is used in all RMPP DATA packets as described in Table 185 Subnet Administration Fields on page 884.

C15-0.1.18: SA **shall** respond to SubnAdmGetTable() and SubnAdmGetTraceTable() requests by performing the Sender role in a receiver-initiated RMPP transmission sequence as described in 13.6.6.1 Receiver-Initiated Transfer on page 790.

- The method used for all packets sent from the Receiver to the Sender **shall** be SubnAdmGetTable() or SubnAdmGetTraceTable(), depending on which initiated the transfer.
- The method used for all packets sent from the Sender to the Receiver **shall** be SubnAdmGetTableResp().
- The attribute contents in all packets except RMPP DATA packets **shall** be ignored following the initial SubnAdmGetTable() or SubnAdmGetTraceTable().
- Transfers initiated by SubnAdmGetTable() **shall** use the Attribute ID present in the initiating SubnAdmGetTable() for all packets of the protocol, unless specified otherwise for the attribute used in the initiating SubnAdmGetTable().
- Transfers initiated by SubnAdmGetTraceTable **shall** use the PathRecord Attribute ID for all packets sent from the Receiver to the Sender, and the TraceRecord Attribute ID for all packets sent from the Sender to the Receiver.

C15-0.1.19: SA **shall** respond to a SubnGetMulti() request by performing a double-sided RMPP transmission sequence as described in 13.6.6.3 Sender-Initiated Double-Sided Transfer on page 792.

- The method used for all packets sent from the Receiver to the Sender **shall** be SubnGetMulti().
- The method used for all packets sent from the Sender to the Receiver **shall** be SubnGetMultiResp().
- The attribute contents in all packets except RMPP DATA packets **shall** be ignored following the initial SubnAdmGetMulti().
- All MADs in the transmission sequence **shall** use the Attribute ID present in the initiating SubnAdmGetMulti().

C15-0.1.20: The initial DATA packet of all RMPP transmission sequences to or from SA **shall** specify exactly the PayloadLength of the data transferred.

Note that this PayloadLength must include not just the lengths of the SubnetAdminData data, but also 20 additional bytes per packet sent to account for the SA headers following the RMPP header (SM_Key, AttributeOffset, Reserved, ComponentMask); and any additional padding between multiple attributes as indicated by the AttributeOffset field.

15.4 OPERATIONS

This section describes the operational aspects of SA.

15.4.1 RESTRICTIONS ON ACCESS

There are two types of access restrictions involved in SA: Authenticating the requester of information, and restricting the data that the requester is allowed to receive. These are discussed below.

The SA access restrictions described here are based on partition membership, and are intended to implement this rule: If access to data is allowed by partition membership, that access is granted; but if it is disallowed, the requester should remain unaware of the existence of that information and of the network elements containing that information. Additionally, in no event is authentication information made available to untrusted requests. That a node's PortInfo:M_Key and PortInfo:M_KeyProtectBits may prohibit access to some data by SMPs without a valid M_Key has no bearing on SA access restrictions.

15.4.1.1 ACCESS RESTRICTIONS FOR PATHRECORDS

C15-0.1.21: Subnet Administration **shall** return to a requester only path attributes for which the source port, destination port, and requester all share a P_Key pairwise. See the remainder of this section (15.4.1.1 Access Restrictions For PathRecords on page 921) for a detailed explanation.

- Two ports share a P_Key when there is at least one valid P_Key in one port's P_Key Table that matches a P_Key in the other port's P_Key Table. (See 10.9.3 Partition Key Matching on page 526 for the definition of P_Key matching, and 10.9.1.2 Special P_Keys on page 524 for the definition of a valid P_Key.)
- "Pairwise" means that P_Key sharing **shall** be present between three pairs of ports: path source port and path destination port; path source port and a port of the requester; path destination port and a port of the requester. Each of the three matches may be based on the matching of different P_Keys.
- The path source and destination ports used to determine sharing are the ones that are implicit in the SGID (or SLID) and DGID (or DLID) of the path.

- All ports involved in this determination **shall** be on the subnet administered by the Subnet Administrator to which the request is directed.

15.4.1.2 ACCESS RESTRICTIONS FOR OTHER ATTRIBUTES

The term “trusted request” used in the following compliance statements refers to a request that contains a valid SM_Key in the SM_Key component of the SA header.

C15-0.1.22: This compliance statement is obsolete and has been replaced by statement C15-0.2.1:.

C15-0.2.1: When a requester node sends a trusted request to SA, the requested data **shall** be returned. When a requester node sends a non-trusted request for data to SA that would provide information about a subject node, the SA **shall** return only data providing information about subject nodes for which the requester shares a P_Key, with exceptions noted below in C15-0.1.23:.

Sharing is defined as follows:

- Two endports share a P_Key when there is at least one valid P_Key in one endport's P_Key Table that matches a P_Key in the other endport's P_Key Table. (See 10.9.3 Partition Key Matching on page 526 for the definition of P_Key matching, and 10.9.1.2 Special P_Keys on page 524 for the definition of a valid P_Key.)
- All endports involved in this determination must be on the subnet administered by the Subnet Administrator to which the request is directed.

As an example of a case where SA might return data that provides information about other nodes that is covered by C15-0.1.22:, consider Table 203 ServiceRecord on page 895. That attribute contains a ServiceGID component that identifies an endport. C15-0.1.22: implies that SA does not return a ServiceRecord containing a ServiceGID which identifies an endport that does not share a P_Key with the endport to which the ServiceRecord is returned.

C15-0.1.23: This compliance statement is obsolete and has been replaced by statement C15-0.2.2:.

C15-0.2.2: Subnet Administration **shall** follow the following additional rules concerning data access:

- PortInfoRecords **shall** always be provided with the M_Key component set to 0, except in the case of a trusted request, in which case the actual M_Key component contents **shall** be provided.
- P_KeyTableRecords and ServiceAssociationRecords **shall** only be provided in responses to trusted requests.

- ServiceRecords **shall** always be provided with the ServiceKey component set to 0, except in the case of a trusted request, in which case the actual ServiceKey component contents **shall** be provided.
- InformInfoRecords **shall** always be provided with the QPN set to 0, except for the case of a trusted request, in which case the actual subscriber QPN **shall** be returned.
- MCMemberRecords **shall** always be provided with the PortGID, JoinState and ProxyJoin components set to 0, except for the case of a trusted request, in which case the actual component contents **shall** be provided.
- TraceRecords **shall** be returned to any endnode requesting them.
- SubnAdmSet(InformInfo) subscriptions for SM security traps (see Table 131 on page 812) **shall** be provided only if they come from a trusted source.

15.4.2 LOCATING SUBNET ADMINISTRATION

C15-0.1.24: It **shall** be possible to determine the location of SA from any endport by sending a GMP to QP1 (the GSI) of the node identified by the endport's PortInfo:MasterSMLID, using in the GMP the base LID of the endport as the SLID, the endport's PortInfo:MasterSMSL as the SL, the well-known Q_Key (0x8001_0000), and whichever of the default P_Keys (0xFFFF or 0x7FFF) was placed in the endport's P_Key Table by the SM (Table 183 Initialization on page 868).

C15-0.1.25: A SubnAdmGet(ClassPortInfo) sent according to C15-0.1.24: **shall** return all information needed to communicate with Subnet Administration. Alternatively, valid GMPs for SA sent according to C15-0.1.24: **shall** either return redirection responses providing all such information, or **shall** be normally processed by SA.

15.4.3 EVENT FORWARDING SUBSYSTEM

The set of traps or notices that may be reported by SA are described in 14.2.5.1 Notices and Traps on page 812. Note: In sending Notices, SA must abide by access restrictions; see o13-14.1.1: on page 746 and C15-0.2.2: on page 922.

C15-0.1.26: Event forwarding operations directed at SA **shall** conform to the common methods as described in 13.4.5 Management Class Methods on page 721.

15.4.4 ADMINISTRATION QUERY SUBSYSTEM

In the administration query subsystem the 64 bit ComponentMask in the SA MAD is used in query operations to specify particular attribute components to query. The ComponentMask can refer to only an entire component, not elements or parts of a component.

C15-0.1.27: For query operations, bit 0 of the ComponentMask **shall** refer to the first component of the SA attribute, bit 1 **shall** refer to the second component, and so forth. For purposes of this bit numbering, “component” **shall** mean either:

- a named attribute component as defined by the attribute tables in this chapter, including rows in which the “Component” column is named “Reserved,” but not including rows solely referring to attribute components defined by tables in other chapters (14.2.5.3 NodeInfo on page 818 is often referenced); call the latter rows “reference rows.”
- a named component of attributes defined by the table in another chapter that is directly referred to by a reference row; if multiple such components are referenced in the same SA attribute, the numbering proceeds sequentially across them.

In other words, the bit numbering used in ComponentMask acts as if SA reference rows are replaced by the attributes they reference, as if a textual macro expansion had been performed (to just one level of embedding).

For example, in querying a PortInfoRecord (see 15.2.5.3 PortInfoRecord on page 891), bit 7 of the ComponentMask refers to PortInfo:CapabilityMask (see 14.2.5.6 PortInfo on page 821). As another example, in querying a LinearForwardingTableRecord (see 15.2.5.6 LinearForwardingTableRecord on page 892), ComponentMask bit 3 identifies an entire LinearForwardingTableBlock (see Table 152 LinearForwardingTable on page 837), but no ComponentMask bits correspond to individual Port Block Elements (see Table 153 Port Block Element on page 837). Also, this chapter's table entry in LinearForwardingTableRecord named “LinearForwardingTable” does not itself use a bit position.

C15-0.1.28: This compliance statement is obsolete and is replaced by statement C15-0.2.3:.

C15-0.2.3: The rules for matching using the ComponentMask are as follows:

- When a ComponentMask bit is 1 in a query, the corresponding component in all responses **shall** be bitwise identical to the value provided in the query.
- When a ComponentMask bit is 0 in a query, the value of the corresponding component **shall not** be considered in choosing responses. This is called “wildcarding,” and a component whose ComponentMask bit is 0 is said to have been “wildcarded.”
- Exceptions:

- Reserved fields **shall not** be considered in choosing responses to a query.
- Particular attribute components may specify matching rules that do not correspond to the above (for example, PathRecord:MtuSelector).

ComponentMask matching is used in read, set, or delete operations. A ComponentMask of all ones means that all components are used for a query. The result of using a ComponentMask of all zeros in a query request is that all attributes of the queried attribute type are returned, except for those not allowed according to 15.4.1 Restrictions on Access on page 921. For some attributes, e.g., PathRecords, some bits of the ComponentMask must be set to 1 in query operations.

For the event forwarding subsystem the ComponentMask is unused.

15.4.5 SUBNADMGETTABLE() / SUBNADMGETTABLERESP()

SubnAdmGetTable() is used to request an attribute table. Operations are allowed on a specific table only, specified by attribute identifier.

An example SubnAdmGetTable() query for PathRecords is shown in Table 206 Example PathRecord Request MAD Header Fields on page 904. An example query for NodeRecords with a NodeType of 3 is shown in Table 214 on page 925; as with Table 206, only header fields whose values differ from their defaults are shown:

Table 214 SubnAdmGetTable query for all NodeRecords with a specific NodeType

Component	Value	Interpretation
MADHeader:MgmtClass	0x03	Specifying SubnAdm class
MADHeader:Method	0x12	Specifying SubnAdmGetTable()
MADHeader:TransactionID	0x0000000011223344	Transaction ID (to be returned in the response)
MADHeader:AttributeID	0x0011	Specifying NodeRecord
RMPPHeader:RMPPFlags.Active	0	Information flags:RMPP is not active
SAHeader:SM_Key	0	SM_Key value not needed for this query
SAHeader:ComponentMask	0x0000000000000010	Selecting those components to be used; only bit 4 is needed.
NodeRecord:NodeType	3	Obtain all NodeRecords with NodeType value of 3, i.e., all routers.

Subnet Administration uses the SubnAdmGetTableResp() to respond to all SubnAdmGetTable() queries.

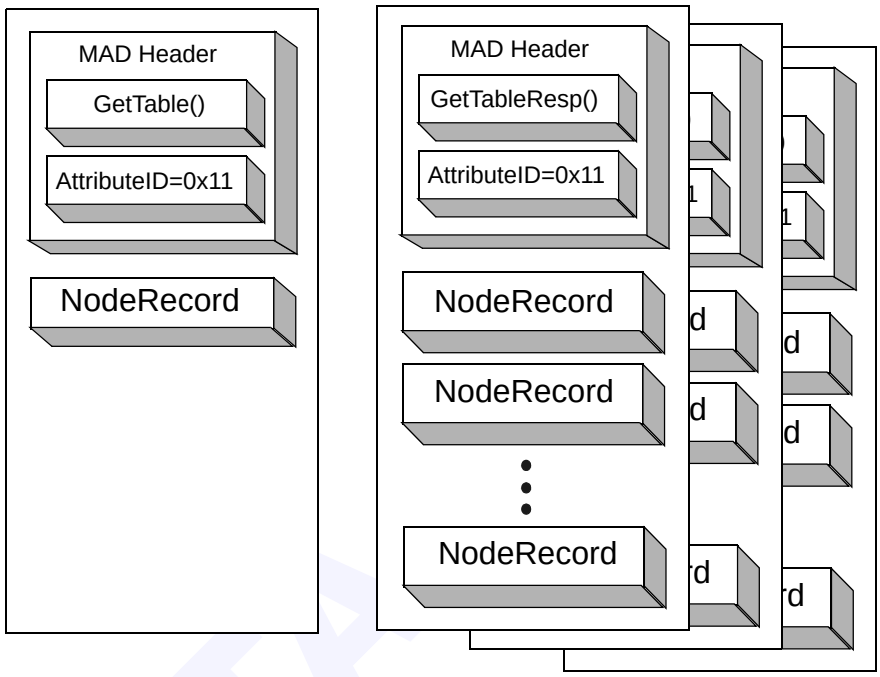


Figure 197 SubnAdmGetTable() Example

C15-0.1.29: SA **shall** indicate a refused request by returning a SubnAdmGetTableResp() with the status field providing the reason for refusal.

A SubnAdmGetTable() and the corresponding SubnAdmGetTableResp() is illustrated schematically in Figure 197 SubnAdmGetTable() Example on page 926. Subsequent SubnAdmGetTableResp() MADs of this transaction will have the RMPP header entry values set to indicate place in the data stream (see 13.6.2 RMPP Packet Formats on page 772). The continuation of the data is to be reassembled by the requester in its own data area.

The attributes for a SubnAdmGetTableResp() for the example of Figure 197 SubnAdmGetTable() Example on page 926 are contained within a SA MAD as depicted. Note that attributes may be broken across successive response MADs.

15.4.6 SUBNADMGET() / SUBNADMGETRESP(): GET AN ATTRIBUTE

C15-0.1.30: Ine response to a SubnAdmGet(), if a single attribute would be returned based on the access rules specified in 15.4.1 Restrictions on Access on page 921 and the matching of components specified by the

ComponentMask, then SubAdmGetResp() **shall** return that attribute with a zero status value.

C15-0.1.31: If SubnAdmGet() fails to satisfy C15-0.1.30:, SubnAdmGetResp() **shall** return with the status field providing the reason for failure (see Table 188 SA MAD Class-Specific Status Encodings on page 886).

Table 215 SubnAdmGet() query for a Particular NodeRecord on page 927 shows an example SubnAdmGet() query which obtains a NodeRecord given its PortGUID; as with Table 206 on page 904, only header fields whose values differ from their defaults are shown:.

Table 215 SubnAdmGet() query for a Particular NodeRecord

Component	Value	Interpretation
MADHeader:MgmtClass	0x03	Specifying SubnAdm class
MADHeader:Method	0x01	Specifying SubnAdmGet()
MADHeader:TransactionID	0x0000000011223344	Transaction ID (to be returned in the response)
MADHeader:AttributeID	0x0011	Specifying NodeRecord
RMPPHeader:RMPPFlags.Active	0	Information flags:RMPP is not active
SAHeader:SM_Key	0	SM_Key value not needed for this query
SAHeader:ComponentMask	0x0000000000000080	Selecting those components to be used; only bit 7 is needed.
NodeRecord:PortGUID	0x0000000000000004	Obtain NodeRecord with given PortGUID.

15.4.7 SUBNADMSET(): SET AN ATTRIBUTE

C15-0.1.32: In response to a SubnAdmSet(), if the attribute contained in that MAD is added to SA, a SubAdmGetResp() **shall** be returned containing the added attribute. Rules for adding an attribute of a given type are specified in the corresponding attribute definition sections.

Note that attributes for which SubnAdmSet() is not supported (Table 190 Subnet Administration Attribute / Method Map on page 890) implicitly cannot be added.

C15-0.1.33: If SubnAdmSet() fails to satisfy C15-0.1.32:, SubnAdmGetResp() **shall** return with the status field providing the reason for failure (see Table 188 SA MAD Class-Specific Status Encodings on page 886).

15.4.8 SUBNADMDELETE(): DELETE AN ATTRIBUTE

The SubnAdmDelete() method provides a means for deleting an attribute that had been added to SA using SubnAdmSet(). Table 190 Subnet Administration Attribute / Method Map on page 890 specifies the attributes

for which SubnAdmDelete() is supported. Note that attributes for which SubnAdmDelete() is not supported cannot be deleted. Rules for deleting an attribute are specified in the corresponding attribute definition sections.

15.4.9 SUBNADMGETTRACETABLE(): TRACE A PATH

The SubnAdmGetTraceTable() method, with SubnAdmGetTableResp(), provides a means for comparing paths provided by SA in order to determine in detail the degree to which they use the same nodes, ports, links, or chassis. It may be used, for example, as a way to check or select a desired degree of failure independence among a set of paths.

The PathRecord given to SubnAdmGetTraceTable() must be completely specified and must specify an existing path that the requester is allowed to access.

o15-0.1.19: This compliance statement is obsolete and has been replaced by statement o15-0.2.8:.

o15-0.2.8: If ClassPortInfo:CapabilityMask.IsMultiPathSupported is 1 for SA, then SA **shall** reject as invalid any SubnAdmGetTraceTable() request which contains a PathRecord that does not describe a valid path; or has a multicast GID or LID as a source or destination component; or does not describe a path the requester is allowed to access according to the rules specified in 15.4.1.1 Access Restrictions For PathRecords on page 921; or does not have all of its components specified, i.e., any bit of the ComponentMask referring to a PathRecord component is 0. In any of these cases, SA **shall** respond with a SubnAdmGetTableResp() indicating a zero-length table and a status code indicating an ERR_REQ_INVALID. (See Table 188 SA MAD Class-Specific Status Encodings on page 886.)

Given a valid PathRecord in a SubnAdmGetTraceTable() request that describes an existing path, SA uses SubnAdmGetTableResp() to return the path's elements as a table of TraceRecords, starting at the source and terminating at the destination.

o15-0.1.20: This compliance statement is obsolete and has been replaced by statement o15-0.2.9:.

o15-0.2.9: If ClassPortInfo:CapabilityMask.IsMultiPathSupported is 1 for SA, then if SubnAdmGetTraceTable() is sent to SA with a PathRecord valid for SubnAdmGetTraceTable() (see o15-0.1.19:), then SA **shall** respond with a SubnAdmGetTableResp() method that returns a table of TraceRecord attributes. The TraceRecords of that table **shall** describe every node traversed by the path sent to SA, and how that path traverses each node. The order of TraceRecords in that table **shall** be the order that a packet sent on that path would traverse those nodes, starting with the path's source node and ending with its destination node.

15.4.10 SUBNADMGETMULTI() / SUBNADMGETMULTIRESP(): SEND & RECEIVE MULTIPLE PACKETS

SubnAdmGetMulti() is used when the reliable multipacket protocol (RMPP) is used to send an attribute or attributes to SA from a client. SubnAdmGetMultiResp() then, in response, uses RMPP to send an attribute or attributes from SA to the client. In contrast, for example, RMPP is used by SubnAdmGetTable() only to send attributes from SA to the client.

CHAPTER 16: GENERAL SERVICES

This chapter describes the range of management services that the IBA provides under general services, except for the Subnet Administration which is described in the previous chapter. General management services provide the following management classes:

- Performance Management - provides methods that enable a manager to retrieve performance statistics and error information from IBA components.
- Baseboard Management - provides a means to transport messages to components beyond the subnet, to “out of band” components. An example might be to chassis temperature monitoring and control hardware on an IBA channel adapter.
- Device Management - provides the means to perform I/O controller / I/O unit management. This class defines the mechanisms to send and receive device management packets between two subnet-attached points, typically between an HCA and a TCA. The TCA provides an interface to the I/O controller and I/O device.
- SNMP Tunneling - provides a set of methods, data formats and attributes to support SNMP tunneling. The SNMP packet is embedded in the IBA-compliant management datagram.
- Vendor Specific - provides a set of general purpose methods. Vendors are free to define new methods and attributes, however they must conform to management datagram formats and restrictions described herein.
- Application Specific - provides a set of general purpose methods. Applications are free to define new methods and attributes, however they must conform to management datagram formats and restrictions described herein.
- Communication Management - provides the mechanisms to establish, terminate, and migrate connections between nodes, and provides basic service ID resolution.

16.1 PERFORMANCE MANAGEMENT

C16-1: The Performance Management Agent is mandatory on all nodes.

The Performance Management class provides mechanisms to enable a performance management entity to retrieve performance and error statistics from InfiniBand components. Performance quantities are divided into two classes:

- Mandatory for all ports of all nodes (TCAs, HCAs, Switches, and Routers). These quantities are deemed necessary to support fundamental instrumentation and performance analysis of a multi-vendor InfiniBand fabric
- Optional. These quantities may be implemented at the vendor's discretion, and are described here as an aid to standardization.

16.1.1 MAD FORMAT

C16-2: The datagrams in the Performance class **shall** conform to the MAD format and use as specified in 13.4 Management Datagrams on page 717 and further customized in Figure 198 Performance Management MAD Format on page 931 and Table 216 Performance Management MAD Fields on page 931 below.

bytes	bits 31-24	bits 23-16	bits 15-8	bits 7-0
0	Common MAD Header			
...				
20				
24	Reserved			
...				
60				
64	Data			
...				
252				

Figure 198 Performance Management MAD Format

Table 216 Performance Management MAD Fields

Field Name	Length	Description
Common MAD Header	24 bytes	Common MAD Header as described in 13.4.2 Management Datagram Format on page 718
Reserved	40 bytes	Reserved.
Data	192 bytes	Attribute data is mapped bit for bit from the format described in the following sections to the start of this data field. If the attribute is smaller than the data field, the content of the remainder of the data field is unspecified.

16.1.1.1 STATUS FIELD

The Status field is described in 13.4.7 Status Field on page 731. No class-specific bits are defined.

Table 217 Performance Management Status Field

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.7 Status Field on page 731
8-15	-	Class-specific bits are reserved

16.1.2 METHODS

The Performance Management class uses a subset of the common methods described in 13.4.5 Management Class Methods on page 721.

C16-2.1.1: A Performance Management Agent **shall** support the methods listed in Table 218 Performance Management Methods on page 932. All method type values not listed in the Table are reserved.

Table 218 Performance Management Methods

Method Type	Value	Description
PerformanceGet()	0x01	Request a get (read) of a class specific information attribute
PerformanceSet()	0x02	Request a set (write) of a class specific information attribute.
PerformanceGetResp()	0x81	Response from a Get() or Set() request.

16.1.3 MANDATORY ATTRIBUTES

C16-2.1.2: A Performance Management Agent **shall** support the mandatory attributes listed in Table 219 Mandatory Performance Management Attributes on page 932 and Table 220 Mandatory Performance Management Attribute / Method Map on page 933. All attribute IDs not listed in Table 219: Mandatory Performance Management Attributes and Table 226 Optional Performance Management Attributes on page 950 are reserved.

The attributes are described in detail in the sections following the table. The use model for these attributes is described in 16.1.3.6 Typical Performance Attribute Use Model on page 949.

Table 219 Mandatory Performance Management Attributes

Attribute Name	Attribute ID	AttributeModifier	Description
ClassPortInfo	0x0001	0x00000000	See 13.4.8.1 ClassPortInfo on page 734

Table 219 Mandatory Performance Management Attributes (Continued)

Attribute Name	Attribute ID	AttributeModifier	Description
PortSamplesControl	0x0010	Selects one of n independent sampling mechanisms; zero (0) **shall** be implemented.	Port Performance Data Sampling Control. See 16.1.3.2 PortSamplesControl on page 933
PortSamplesResult	0x0011	Selects one of n independent sampling mechanisms; zero (0) **shall** be implemented.	Port Performance Data Sampling Results. See 16.1.3.4 PortSamplesResult on page 944
PortCounters	0x0012	0x00000000	Port Basic Performance & Error Counters. See 16.1.3.5 PortCounters on page 945
Reserved	0x0013-0x0014	0x00000000-0xFFFFFFFF	Reserved

Table 220 Mandatory Performance Management Attribute / Method Map

Attribute Name	PerformanceGet()	PerformanceSet()
ClassPortInfo	X	
PortSamplesControl	X	X
PortSamplesResult	X	
PortCounters	X	X

16.1.3.1 CLASSPORTINFO

The ClassPortInfo attribute is described in 13.4.8.1 ClassPortInfo on page 734.

Table 221 Performance Management ClassPortInfo:CapabilityMask

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.8.1 ClassPortInfo on page 734
8	AllPortSelect	If reported as 1, indicates that all attributes containing the PortSelect component support setting it to 0xFF to gather data from all ports at once. If reported as 0, using 0xFF in PortSelect results in undefined behavior.
9-15	-	Class-specific bits are reserved

16.1.3.2 PORTSAMPLESCONTROL

The PortSamplesControl attribute is mandatory. It provides a means of initiating a sample and selecting, for one selected port during the specified interval, quantities to be sampled such as:

- The amount of data sent and received
- The number of packets sent and received
- The transmit queue depth at the start of the interval

The complete list of quantities that can be sampled using this mechanism is given in Table 223 CounterSelect Values on page 941.

Sampling is initiated by means of a PerformanceSet(PortSamplesControl). Sampling status and results are obtained by means of a PerformanceGet(PortSamplesResult).

To support random sampling that is decoupled from MAD latencies and other port activities at either the sender or receiver, the PortSamplesControl attribute provides a means to specify a delayed start time for the sample interval. See the SampleStart component in Table 222 PortSamplesControl on page 934.

Performance sampling operations are based on a standard time interval called a tick. A tick is a multiple of the link transfer period. For example, a multiple of 400 picoseconds for a link running at 2.5 giga-transfers per second. Implementers are given a range of multipliers to choose from.

The AttributeModifier selects one of several possible independent sampling mechanisms.

C16-3: All nodes **shall** implement PortSamplesControl and PortSamplesResult corresponding to an AttributeModifier of zero. Implementation of additional sets of PortSamplesControl and PortSamplesResult permits simultaneous sampling of multiple ports, and **shall** use ascending AttributeModifier values starting with one (1). The number of additional sets implemented is defined in PortSamplesControl.SampleMechanisms.

C16-4: This compliance statement is obsolete and has been replaced by C16-4.1.1:.

C16-4.1.1: For each sampling mechanism, at least one and up to 15 counters **shall** be implemented.

Table 222 PortSamplesControl

Component	Access	Length (bits)	Offset (bits)	Description
OpCode	RW	8	0	Used to select a specific packet op code (as found in BTH) when sampling optional quantities that are op code specific. If OpCode is 0xFF, all op codes are sampled as one otherwise only one op code can be sampled at a time, although multiple quantities can be sampled for the same op code.

Table 222 PortSamplesControl (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
PortSelect	RW	8	8	Selects the port that will be sampled. For a channel adapter or router, PortSelect refers to an endport with the only valid value being the port number of which the request was received. For a switch, PortSelect refers to a switch port with valid values ranging from 0 to the number of ports in the switch. However, 0 is only valid for the enhanced switch management port; it is ignored for the base management port. If gathering data from all ports at once is supported (see Table 221 Performance Management ClassPortInfo:CapabilityMask on page 933), setting PortSelect to 0xFF will cause samples from all valid ports (as defined in the previous paragraph) to be accumulated. When selecting invalid port values, any results are undefined.
Tick	RO	8	16	Indicates the node's sampling clock interval as a multiple of 10x the link transfer period. For a 2.5 Gtransfer link, the transfer period is 400 picoseconds. The encoding is: 0x00 = 10 x link transfer period (4 nanoseconds for a 2.5 Gtransfer link) 0x01 = 20 x link transfer period 0x02 = 30 x link transfer period ... 0xFF = 2560 x link transfer period To maximize utility of the performance attributes, implementers are encouraged to choose the smallest practical tick size
Reserved	RO	5	24	Reserved
CounterWidth	RO	3	29	Indicates the actual width in bits of the following components: • SampleStart • SampleInterval • PortSamplesResult:Counter0 to 14 The encoding is: 0 = 16 bits 1 = 20 bits 2 = 24 bits 3 = 28 bits 4 = 32 bits 5-7 = reserved Counters smaller than 32 bits **shall** be implemented as the least significant bits of the corresponding 32-bit attribute component, with the unimplemented upper bits of the component returning zeroes for Get and ignored for Set.
Reserved	RO	2	32	Reserved

Table 222 PortSamplesControl (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
CounterMask0	RO	3	34	A bitmask that determines the capabilities of PortSamplesResult:Counter0. Bit 0 = supports all mandatory quantities; **shall** be 1 Bit 1 = supports optional quantities Bit 2 = supports vendor-defined quantities
CounterMasks1to9	RO	27	37	An array of nine 3-bit bitmasks, each of which determines the capabilities of an optional counter in PortSamplesResult. The most significant 3-bit field corresponds to PortSamplesResult:Counter1; the least significant field corresponds to PortSamplesResult:Counter9 Encoding: Bit 0 = supports all mandatory quantities Bit 1 = supports optional quantities Bit 2 = supports vendor-defined quantities All bits zero means the counter is not implemented
Reserved	RO	1	64	Reserved.
CounterMasks10to14	RO	15	65	An array of five 3-bit bitmasks, each of which determines the capabilities of an optional counter in PortSamplesResult. The most significant 3-bit field corresponds to PortSamplesResult:Counter10; the least significant field corresponds to PortSamplesResult:Counter14 Encoding: Bit 0 = supports all mandatory quantities Bit 1 = supports optional quantities Bit 2 = supports vendor-defined quantities All bits zero means the counter is not implemented
SampleMechanisms	RO	8	80	The number of independent sample mechanisms implemented (i.e., sets of PortSamplesControl and PortSamplesResult), minus one: 0 = one sample mechanism is available (addressed via AttributeModifier zero) 1 = two sample mechanisms are available, AttributeModifiers 0 and 1 ... 255 = 256 sample mechanisms are available, addressed via AttributeModifiers 0 through 255 Providing multiple sampling mechanisms is optional. N sample mechanisms would permit N independent samples to be run simultaneously. The result of using an AttributeModifier value of 0xFFFFFFFF is vendor-specific.
Reserved	RO	6	88	Reserved

Table 222 PortSamplesControl (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
SampleStatus	RO	2	94	Indicates the status of sampling: 0 = sampling is complete and the results are available from the PortSamplesResult attribute 1 = the SampleStart timer is running. All sample counter values in PortSamplesResult are undefined 2 = sampling is underway. All sample counter values in PortSamplesResult are undefined 3 = reserved While SampleStatus is non-zero, a PerformanceSet(PortSamplesControl) will not affect PortSamplesControl and will return the existing values of all components

Table 222 PortSamplesControl (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
OptionMask	RO	64	96	A bit mask indicating which optional InfiniBand performance quantities are implemented, if any. See Table 223 CounterSelect Values on page 941 for a description of each quantity or set of quantities: Bit 0 (LSB) = reserved. Bit 1 = PortXmitQueue[n] Bit 2 = PortXmitDataVL[n] Bit 3 = PortRcvDataVL[n] Bit 4 = PortXmitPktVL[n] Bit 5 = PortRcvPktVL[n] Bit 6 = PortRcvErrorDetails:PortLocalPhysicalErrors Bit 7 = PortRcvErrorDetails:PortMalformedPacketErrors Bit 8 = PortRcvErrorDetails:PortBufferOverrunErrors Bit 9 = PortRcvErrorDetails:PortDLIDMappingErrors Bit 10 = PortRcvErrorDetails:PortVLMappingErrors Bit 11 = PortRcvErrorDetails:PortLoopingErrors Bit 12 = PortXmitDiscardDetails:PortInactiveDiscards Bit 13 = PortXmitDiscardDetails:PortNeighborMTUDiscards Bit 14 = PortXmitDiscardDetails:PortSwLifetimeLimitDiscards Bit 15 = PortXmitDiscardDetails:PortSwHOQLifetimeLimitDiscards Bit 16 = PortOpRcvCounters:PortOpRcvPkts Bit 17 = PortOpRcvCounters:PortOpRcvData Bit 18 = PortFlowCtlCounters:PortXmitFlowPkts Bit 19 = PortFlowCtlCounters:PortRcvFlowPkts Bit 20 = PortVLOpPackets:PortVLOpPackets[n] Bit 21 = PortVLOpData:PortVLOpData[n] Bit 22 = PortVLXmitFlowCtlUpdateErrors:PortVLXmitFlowCtlUpdateErrors[n] Bit 23 = PortVLXmitWaitCounters:PortVLXmitWait[n] Bits 24 - 47 Reserved Bit 48 = SwPortVLCongestion:SWPortVLCongestion[n] Bit 49 = PortRcvConCtrl:PortPktRcvFECN Bit 50 = PortRcvConCtrl:PortPktRcvBECN Bit 51 = PortSLRcvFECN:PortSLPktRcvFECN[n] Bit 52 = PortSLRcvBECN:PortSLPktRcvBECN[n] Bit 53 = PortXmitConCtrl:PortXmitTimeCong Bit 54 = PortVLXmitTimeCong:PortVLXmitTimeCong[n] Bits 55 - 63 Reserved Performance quantities that are counted per VL are limited to the actual number of VLs implemented. The result of selecting an unimplemented quantity is all zeroes. See Annex <<ref to Congestion Control Annex>> for descriptions of the counters designated by bits 49 - 54.

Table 222 PortSamplesControl (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
VendorMask	RO	64	160	A bitmask indicating which vendor-specific counters are implemented. **Shall** be zero if the node does not support any vendor-specific counters, otherwise use is vendor-defined
SampleStart	RW	32	224	Determines when the sampling interval starts. When Set, this value is loaded into a timer and the following events occur: • SampleStatus is set to 1 • Counters in PortSamplesResult are set to zero • The timer begins decrementing once per tick When the timer reaches zero, timing stops and the following events occur • The PortXmitQueue quantities if selected are latched • PortSamplesResult counters are started • SampleStatus is set to 2 • The SampleInterval timer is started The SampleStart timer allows a performance application to randomize the sample start time and insure decoupling from node or network events. Values used will typically be 10's of milliseconds. It is the fine granularity of this interval with respect to the link rate that makes decoupling possible
SampleInterval	RW	32	256	Determines the length of the sampling interval. When Set, this value is loaded into a timer. When the SampleStart counter reaches zero, this timer begins decrementing once per tick. When it reaches zero, timing stops and the following events occur: • PortSamplesResult counters are stopped and the resulting values made available • SampleStatus is set to zero
Tag	RW	16	288	Used by a performance application when it does a PerformanceSet(PortSamplesControl) to uniquely identify its sample run in case of a collision with another performance application When an application wishes to start a sample run, it should pick a random Tag value and do a PerformanceSet(PortSamplesControl). If the returned value of Tag does not match the selected value, another application is using the sampling mechanism. In this case the first application should wait for a suitable time and retry its sample
CounterSelect0	RW	16	304	Selects quantity to be sampled by PortSamplesResult:Counter0 as defined in Table 223 CounterSelect Values on page 941. If an unimplemented quantity is selected, a Get to PortSamplesResult:Counter0 returns zeroes
CounterSelect1	RW	16	320	Similar to CounterSelect0; selects quantity to be sampled by PortSamplesResult:Counter1
CounterSelect2	RW	16	336	Similar to CounterSelect0; selects quantity to be sampled by PortSamplesResult:Counter2
CounterSelect3	RW	16	352	Similar to CounterSelect0; selects quantity to be sampled by PortSamplesResult:Counter3

Table 222 PortSamplesControl (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
CounterSelect4	RW	16	368	Similar to CounterSelect0; selects quantity to be sampled by PortSamplesResult:Counter4
CounterSelect5	RW	16	384	Similar to CounterSelect0; selects quantity to be sampled by PortSamplesResult:Counter5
CounterSelect6	RW	16	400	Similar to CounterSelect0; selects quantity to be sampled by PortSamplesResult:Counter6
CounterSelect7	RW	16	416	Similar to CounterSelect0; selects quantity to be sampled by PortSamplesResult:Counter7
CounterSelect8	RW	16	432	Similar to CounterSelect0; selects quantity to be sampled by PortSamplesResult:Counter8
CounterSelect9	RW	16	448	Similar to CounterSelect0; selects quantity to be sampled by PortSamplesResult:Counter9
CounterSelect10	RW	16	464	Similar to CounterSelect0; selects quantity to be sampled by PortSamplesResult:Counter10
CounterSelect11	RW	16	480	Similar to CounterSelect0; selects quantity to be sampled by PortSamplesResult:Counter11
CounterSelect12	RW	16	496	Similar to CounterSelect0; selects quantity to be sampled by PortSamplesResult:Counter12
CounterSelect13	RW	16	512	Similar to CounterSelect0; selects quantity to be sampled by PortSamplesResult:Counter13
CounterSelect14	RW	16	528	Similar to CounterSelect0; selects quantity to be sampled by PortSamplesResult:Counter14

16.1.3.3 COUNTERSELECT VALUES

Table 223 CounterSelect Values on page 941 lists the values that can be used in the CounterSelect[n] components of the PortSamplesControl attribute to select a particular quantity to sample.

Quantities that can be sampled are divided into 3 ranges:

- Mandatory quantities (0x0000 - 0x3FFF).

C16-5: Mandatory quantities for performance sampling **shall** be implemented on ports as specified by PortSamplesControl:PortSelect.

- Optional quantities (0x4000 - 0xBFFF).

o16-1: If provided, optional quantities for performance sampling **shall** be implemented as described.

- Vendor quantities (0xC000 - 0xFFFF). Vendors may define and implement their own quantities in this range

Table 223 CounterSelect Values

Sample Select Value	Name	Description
Mandatory Quantities		
0x0000	Reserved	Reserved
0x0001	PortXmitData	Total number of data octets, divided by 4, transmitted on all VLs during the sampling interval from the port selected by PortSelect. This includes all octets between (and not including) the start of packet delimiter and the VCRC, and may include packets containing errors (see Figure 51 Packet Receiver State Machine on page 174 and Figure 52 Data Packet Check machine on page 176). It excludes all link packets. Implementers may choose to count data octets in groups larger than four but are encouraged to choose the smallest group possible. Results are still reported in units of four octets.
0x0002	PortRcvData	Total number of data octets, divided by 4, received on all VLs during the sampling interval at the port selected by PortSelect. This includes all octets between (and not including) the start of packet delimiter and the VCRC, and may include packets containing errors (see Figure 51 Packet Receiver State Machine on page 174 and Figure 52 Data Packet Check machine on page 176). It excludes all link packets. Implementers may choose to count data octets in groups larger than four but are encouraged to choose the smallest group possible. Results are still reported in units of four octets.
0x0003	PortXmitPkts	Total number of packets transmitted on all VLs from this port. This may include packets with errors (see Figure 51 Packet Receiver State Machine on page 174 and Figure 52 Data Packet Check machine on page 176), and excludes link packets.
0x0004	PortRcvPkts	Total number of packets, including packets containing errors (see Figure 51 Packet Receiver State Machine on page 174 and Figure 52 Data Packet Check machine on page 176) and excluding link packets, received on all VLs during the sampling interval on the port selected by PortSelect.
0x0005	PortXmitWait	The number of ticks during which the port selected by PortSelect had data to transmit but no data was sent during the entire tick either because of insufficient credits or because of lack of arbitration.
0x0006-0x3FFF	Reserved	Reserved.

Table 223 CounterSelect Values (Continued)

Sample Select Value	Name	Description
Optional InfiniBand Quantities		
All quantities that are available in optional attributes as free-running counters are also optionally available for sampling over a given period. Each sampling counter corresponding to an optional running counter is reset to zero for each sample and increments along with the selected running counter during the sampling interval. For certain quantities, such as PortXmitQueue[n], there are no corresponding optional attributes. All values between 0x4000 and 0xBFFF not listed here are reserved and the result of sampling is all zeroes		
0x4n00	PortXmitQueue[n]	Contains the transmit queue depth in bytes on VL “n” of the port selected by PortSelect at the time the SampleStart timer expired The goal of measuring queue depths is to enable software to compute the average time data waits for transmission inside a node. Ideally, a node should increment a counter upon arrival of each byte that is destined for a given output port and should decrement the counter upon departure of each byte from the output port. In practice, this will be impossible to implement precisely. Implementers are encouraged to measure queue depths as accurately as practical and to document any systematic measurement errors. Note that an implementation can compensate for an inherent delay in accounting for arriving bytes by introducing an equal delay in accounting for departing bytes
0x4n01	PortXmitDataVL[n]	Total number of data octets, divided by 4, transmitted on VL “n” from the port selected by PortSelect. This includes all octets between the start of packet and end of packet delimiters. It excludes all control groups and VCRCs. Implementers may choose to count data octets in groups larger than four but are encouraged to choose the smallest group possible. Results are still reported as a multiple of four octets
0x4n02	PortRcvDataVL[n]	Total number of data octets, divided by 4, received on input VL “n” on the port selected by PortSelect. This includes all octets between the start of packet and end of packet delimiters. It excludes all control groups and VCRCs. Implementers may choose to count data octets in groups larger than four but are encouraged to choose the smallest group possible. Results are still reported as a multiple of four octets
0x4n03	PortXmitPktVL[n]	Total number of packets transmitted on VL “n” from the port selected by PortSelect with or without errors.
0x4n04	PortRcvPktVL[n]	Total number of packets received on input VL “n” from the port selected by PortSelect with or without errors.
0x4005	PortRcvErrorDetails:PortLocalPhysicalErrors	See Table 228 PortRcvErrorDetails on page 951.
0x4006	PortRcvErrorDetails:PortMalformedPacketErrors	See Table 228 PortRcvErrorDetails on page 951.

Table 223 CounterSelect Values (Continued)

Sample Select Value	Name	Description
0x4007	PortRcvErrorDetails:PortBufferOverrunErrors	See Table 228 PortRcvErrorDetails on page 951.
0x4008	PortRcvErrorDetails:PortDLIDMappingErrors	See Table 228 PortRcvErrorDetails on page 951.
0x4009	PortRcvErrorDetails:PortVLMappingErrors	See Table 228 PortRcvErrorDetails on page 951.
0x400A	PortRcvErrorDetails:PortLoopingErrors	See Table 228 PortRcvErrorDetails on page 951.
0x400B	PortXmitDiscardDetails:PortInactiveDiscards	See Table 229 PortXmitDiscardDetails on page 953.
0x400C	PortXmitDiscardDetails:PortNeighborMTUDiscards	See Table 229 PortXmitDiscardDetails on page 953.
0x400D	PortXmitDiscardDetails:PortSwLifetimeLimitDiscards	See Table 229 PortXmitDiscardDetails on page 953.
0x400E	PortXmitDiscardDetails:PortSwHOQLifetimeLimitDiscards	See Table 229 PortXmitDiscardDetails on page 953.
0x400F	PortOpRcvCounters:PortOpRcvPkts	See Table 230 PortOpRcvCounters on page 953. The op code to be sampled is selected by PortSamplesControl:OpCode.
0x4010	PortOpRcvCounters:PortOpRcvData	See Table 230 PortOpRcvCounters on page 953. The op code to be sampled is selected by PortSamplesControl:OpCode.
0x4011	PortFlowCtlCounters:PortXmitFlowPkts	See Table 231 PortFlowCtlCounters on page 954.
0x4012	PortFlowCtlCounters:PortRcvFlowPkts	See Table 231 PortFlowCtlCounters on page 954
0x4n13	PortVLOpPackets:PortVLOpPackets[n]	See Table 232 PortVLOpPackets on page 955. The op code to be sampled is selected by PortSamplesControl:OpCode
0x4n14	PortVLOpData:PortVLOpData[n]	See Table 233 PortVLOpData on page 957. The op code to be sampled is selected by PortSamplesControl:OpCode
0x4n15	PortVLXmitFlowCtlUpdateErrors:PortVLXmitFlowCtlUpdateErrors[n]	See Table 234 PortVLXmitFlowCtlUpdateErrors on page 958.
0x4n16	PortVLXmitWaitCounters:PortVLXmitWait[n]	See Table 235 PortVLXmitWaitCounters on page 960
0x4n30	SwPortVLCongestion:SWPortVLCongestion[n]	See Table 236 SwPortVLCongestion on page 962
Vendor-Defined Quantities Semantics for vendor-defined quantities are not part of this specification.		
0xC000-0xFFFF	Reserved	Reserved for vendor-specific counters

16.1.3.4 PORTSAMPLESRESULT

This mandatory attribute reports the results of a particular sample controlled and initiated via the PortSamplesControl attribute.

Table 224 PortSamplesResult

Component	Access	Length (bits)	Offset (bits)	Description
Tag	RO	16	0	Read-only copy of PortSamplesControl:Tag. The Tag mechanism provides a means for performance applications to detect collisions when using the sampling mechanism. After successfully initiating a sample run, an application should wait until the sample should have completed, then repeat a PerformanceGet(PortSamplesResult) until SampleStatus is zero. If after any Get the Tag value in the result does not match the value set by the application at the start of the run, another application has already started a new sample. In this case the first application should wait for a suitable time and retry its sample
Reserved	RO	14	16	Reserved
SampleStatus	RO	2	30	Read-only copy of PortSamplesControl:SampleStatus. Provided here to minimize traffic while application is polling for sample completion
Counter0	RO	32	32	Mandatory counter. When PortSamplesControl:SampleStatus is zero, contains the result of sampling the quantity selected by PortSamplesControl:CounterSelect0. Undefined when PortSamplesControl:SampleStatus is non-zero. The actual number of valid (least significant) bits in the counter is defined by PortSamplesControl:CounterWidth
Counter1	RO	32	64	Optional counter. All zeroes if not implemented; otherwise similar to Counter0. Contains the result of sampling the quantity selected by PortSamplesControl:CounterSelect1
Counter2	RO	32	96	Similar to Counter1; contains the result of sampling the quantity selected by PortSamplesControl:CounterSelect2
Counter3	RO	32	128	Similar to Counter1; contains the result of sampling the quantity selected by PortSamplesControl:CounterSelect3
Counter4	RO	32	160	Similar to Counter1; contains the result of sampling the quantity selected by PortSamplesControl:CounterSelect4
Counter5	RO	32	192	Similar to Counter1; contains the result of sampling the quantity selected by PortSamplesControl:CounterSelect5
Counter6	RO	32	224	Similar to Counter1; contains the result of sampling the quantity selected by PortSamplesControl:CounterSelect6
Counter7	RO	32	256	Similar to Counter1; contains the result of sampling the quantity selected by PortSamplesControl:CounterSelect7
Counter8	RO	32	288	Similar to Counter1; contains the result of sampling the quantity selected by PortSamplesControl:CounterSelect8

Table 224 PortSamplesResult (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
Counter9	RO	32	320	Similar to Counter1; contains the result of sampling the quantity selected by PortSamplesControl:CounterSelect9
Counter10	RO	32	352	Similar to Counter1; contains the result of sampling the quantity selected by PortSamplesControl:CounterSelect10
Counter11	RO	32	384	Similar to Counter1; contains the result of sampling the quantity selected by PortSamplesControl:CounterSelect11
Counter12	RO	32	416	Similar to Counter1; contains the result of sampling the quantity selected by PortSamplesControl:CounterSelect12
Counter13	RO	32	448	Similar to Counter1; contains the result of sampling the quantity selected by PortSamplesControl:CounterSelect13
Counter14	RO	32	480	Similar to Counter1; contains the result of sampling the quantity selected by PortSamplesControl:CounterSelect14

16.1.3.5 PORTCOUNTERS

C16-6: The PortCounters attribute of the Performance class is mandatory.

PortCounters provides basic performance and exception statistics for a port selected by PortCounters:PortSelect.

C16-7: When initially powered-up or reset, the value of all counters on all ports of a node **shall** be set to zero. During operation, instead of over-flowing, they **shall** stop at all ones. At any time, writing (Set) zero into a counter **shall** cause the counter to be reset to zero.

Note that writing (Set) anything other than zero into a counter results in undefined behavior.

Note that although PortCounters is mandatory, it contains components that are optional.

Table 225 PortCounters

Component	Access	Length (bits)	Offset (bits)	Description
Reserved	RO	8	0	Reserved

Table 225 PortCounters (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
PortSelect	RW	8	8	Selects the port that will be accessed. For a channel adapter or router, PortSelect refers to an endport with the only valid value being the port number on which the request was received. For a switch, PortSelect refers to a switch port with valid values ranging from 0 to the number of ports in the switch. However, 0 is only valid for the enhanced switch management port; it is ignored for the base switch management port. If simultaneous gathering of data from all ports is supported (see Table 221 Performance Management ClassPortInfo:Capability-Mask on page 933), setting PortSelect to 0xFF will cause results from all valid ports (as defined in the previous paragraph) to be accumulated. PerformanceSet(PortCounters) with PortSelect set to 0xFF **shall** alter the contents of the counters selected by CounterSelect for all valid ports. Each component in the Port-Counter Attribute in PerformanceGetResp(PortCounters) that is returned in response to PerformanceGet(PortCounters) or PerformanceSet(PortCounters) with PortSelect set to 0xFF **shall** contain the sum of counter values corresponding to that component of all valid ports. When selecting invalid port values, any results are undefined.
CounterSelect	RW	16	16	When writing (Set), selects which counters are affected by the operation. When reading (Get), this is ignored. Bit 0 - SymbolErrorCounter Bit 1 - LinkErrorRecoveryCounter Bit 2 - LinkDownedCounter Bit 3 - PortRcvErrors Bit 4 - PortRcvRemotePhysicalErrors Bit 5 - PortRcvSwitchRelayErrors Bit 6 - PortXmitDiscards Bit 7 - PortXmitConstraintErrors Bit 8 - PortRcvConstraintErrors Bit 9 - LocalLinkIntegrityErrors Bit 10 - ExcessiveBufferOverrunErrors Bit 11 - VL15Dropped Bit 12 - PortXmitData Bit 13 - PortRcvData Bit 14 - PortXmitPkts Bit 15 - PortRcvPkts
SymbolErrorCounter	RW	16	32	Total number of minor link errors detected on one or more physical lanes. Refer to the InfiniBand Architecture Specification, Volume 2, Link/Phy Interface.
LinkErrorRecovery-Counter	RW	8	48	Total number of times the Port Training state machine has successfully completed the link error recovery process. Refer to the InfiniBand Architecture Specification, Volume 2, Link/Phy Interface.

Table 225 PortCounters (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
LinkDownedCounter	RW	8	56	Total number of times the Port Training state machine has failed the link error recovery process and downed the link. Refer to the InfiniBand Architecture Specification, Volume 2, Link/Phy Interface.
PortRcvErrors	RW	16	64	Total number of packets containing an error that were received on the port. These errors include: • Local physical errors (ICRC, VCRC, FCCRC, and all physical errors that cause entry into the BAD PACKET or BAD PACKET DISCARD states of the packet receiver state machine) • Malformed data packet errors (LVer, length, VL) • Malformed link packet errors (operand, length, VL) • Packets discarded due to buffer overrun
PortRcvRemotePhysicalErrors	RW	16	80	Total number of packets marked with the EBP delimiter received on the port.
PortRcvSwitchRelayErrors	RW	16	96	Total number of packets received on the port that were discarded because they could not be forwarded by the switch relay. Reasons for this include: • DLID mapping (see the description of PortDLIDMappingErrors in Table 228 PortRcvErrorDetails on page 951) • VL mapping • Looping (output port = input port)
PortXmitDiscards	RW	16	112	Total number of outbound packets discarded by the port because the port is down or congested. Reasons for this include: • Output port is not in the active state • Packet length exceeded NeighborMTU • Switch Lifetime Limit exceeded • Switch HOQ Lifetime Limit exceeded This may also include packets discarded while in VLStalled State.
PortXmitConstraintErrors	RW	8	128	Total number of packets not transmitted from the switch physical port for the following reasons: • FilterRawOutbound is true and packet is raw • PartitionEnforcementOutbound is true and packet fails partition key check or IP version check
PortRcvConstraintErrors	RW	8	136	Total number of packets received on the switch physical port that are discarded for the following reasons: • FilterRawInbound is true and packet is raw • PartitionEnforcementInbound is true and packet fails partition key check or IP version check
Reserved	RO	8	144	Reserved

Table 225 PortCounters (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
LocalLinkIntegrityErrors	RW	4	152	The number of times that the count of local physical errors exceeded the threshold specified by LocalPhyErrors; see 7.12.1 Error Detection on page 219 and Table 145 PortInfo on page 822.
ExcessiveBufferOverrunErrors	RW	4	156	The number of times that OverrunErrors consecutive flow control update periods occurred, each having at least one overrun error; see 7.12.1 Error Detection on page 219 and Table 145 PortInfo on page 822.
Reserved	RO	16	160	Reserved
VL15Dropped	RW	16	176	Number of incoming VL15 packets dropped due to resource limitations (e.g., lack of buffers) in the port
PortXmitData	RW	32	192	Optional; **shall** be zero if not implemented. Total number of data octets, divided by 4, transmitted on all VLs from the port. This includes all octets between (and not including) the start of packet delimiter and the VCRC, and may include packets containing errors (see Figure 51 Packet Receiver State Machine on page 174 and Figure 52 Data Packet Check machine on page 176). It excludes all link packets. Implementers may choose to count data octets in groups larger than four but are encouraged to choose the smallest group possible. Results are still reported as a multiple of four octets.
PortRcvData	RW	32	224	Optional; **shall** be zero if not implemented. Total number of data octets, divided by 4, received on all VLs at the port. This includes all octets between (and not including) the start of packet delimiter and the VCRC, and may include packets containing errors (see Figure 51 Packet Receiver State Machine on page 174 and Figure 52 Data Packet Check machine on page 176). It excludes all link packets. When the received packet length exceeds the maximum allowed packet length specified in C7-45:, the counter may include all data octets exceeding this limit. Implementers may choose to count data octets in groups larger than four but are encouraged to choose the smallest group possible. Results are still reported as a multiple of four octets.
PortXmitPkts	RW	32	256	Optional; **shall** be zero if not implemented. Total number of packets transmitted on all VLs from the port. This may include packets with errors (see Figure 51 Packet Receiver State Machine on page 174 and Figure 52 Data Packet Check machine on page 176), and excludes link packets.
PortRcvPkts	RW	32	288	Optional; **shall** be zero if not implemented. Total number of packets, including packets containing errors (see Figure 51 Packet Receiver State Machine on page 174 and Figure 52 Data Packet Check machine on page 176), and excluding link packets, received from all VLs on the port.

16.1.3.6 TYPICAL PERFORMANCE ATTRIBUTE USE MODEL

This section describes how PerformanceGet() and PerformanceSet() can be used to read and reset the performance counters.

PortSamplesControl and PortSamplesResult are used together to sample one or more quantities over a specified period of time:

The application can first determine the node's sampling capabilities via a PerformanceGet(PortSamplesControl). This will return the number and width of available counters, the quantities that can be sampled, and the basic time interval (tick). From these the application can compute the maximum sample interval that will not cause counter overflow.

C16-8: This Compliance Statement is obsolete and has been deleted.

To perform a sampling operation, the performance manager will typically perform the following sequence:

- Select a random value for SampleStart. The SampleStart timer allows a performance application to randomize the sample start time and insure decoupling from node or network events. Values used will typically be 10's of milliseconds.
- Select a random Tag value. This value is used to detect collisions among multiple independent performance applications accessing the same node
- Select a SampleInterval value, the quantities to be sampled, and the counter that will be assigned to count each quantity.
- Do a PerformanceSet(PortSamplesControl). If the returned value of Tag does not match the selected value, another application is using the sampling mechanism. In this case the first application **must** wait for a suitable time and retry the PerformanceSet().
- Once the sample has been successfully started, the application should wait until the SampleStart and SampleInterval timers should have expired, then repeat PerformanceGet(PortSamplesResult) until SampleStatus is zero. If at any time the returned Tag value no longer matches the application's chosen value, regardless of SampleStatus, it means another application has gained control of the sampling mechanism. In this case the first application could restart the sampling process.

If more than one set of sampling mechanisms is implemented, the additional ones are addressed using non-zero AttributeModifier values. The previous use model applies to each pair of PortSamplesControl and PortSamplesResult, treating each as an independent entity.

16.1.4 OPTIONAL ATTRIBUTES

Performance Management defines the optional Attributes and Attribute-Modifier use summarized in Table 226 Optional Performance Management Attributes on page 950. They are described in detail in the sections following the table. All quantities in these attributes can also be sampled via the PortSamplesControl mechanism. The optional Attributes available are reflected by the OptionMask in the PortSamplesControl attribute.

The behavior of all optional counters is as specified in C16-7: on page 945.

o16-2: This optional compliance statement is obsolete and has been deleted.

o16-2.1.1: If any components of an optional Performance Management attribute is supported according to PortSamplesControl:OptionMask, then that attribute **shall** be supported by a Performance Management Agent, as indicated in Table 226 Optional Performance Management Attributes on page 950 and Table 227 Optional Performance Management Attribute / Method Map on page 951.

Table 226 Optional Performance Management Attributes

Attribute Name	Attribute ID	Attribute-Modifier	Description
PortRcvErrorDetails	0x0015	0x00000000	Port Detailed Error Counters. See 16.1.4.1 PortRcvErrorDetails on page 951.
PortXmitDiscardDetails	0x0016	0x00000000	Port Transmit Discard Counters. See 16.1.4.2 PortXmitDiscardDetails on page 953.
PortOpRcvCounters	0x0017	0x00000000	Port Receive Counters per Op Code. See 16.1.4.3 PortOpRcvCounters on page 953.
PortFlowCtlCounters	0x0018	0x00000000	Port Flow Control Counters. See 16.1.4.4 PortFlowCtlCounters on page 954.
PortVLOpPackets	0x0019	0x00000000	Port Packets Received per Op Code per VL. See 16.1.4.5 PortVLOpPackets on page 955.
PortVLOpData	0x001A	0x00000000	Port Data Received per Op Code per VL. See 16.1.4.6 PortVLOpData on page 957.
PortVLXmitFlowCtlUpdateErrors	0x001B	0x00000000	Port Flow Control update errors per VL. See 16.1.4.7 PortVLXmitFlowCtlUpdateErrors on page 958
PortVLXmitWaitCounters	0x001C	0x00000000	Port Ticks Waiting to Transmit Counters per VL. See 16.1.4.8 PortVLXmitWaitCounters on page 960.
PortCountersExtended	0x001D	0x00000000	Extended Port Basic Performance & Error Counters. See 16.1.4.11 PortCountersExtended on page 965

Table 226 Optional Performance Management Attributes (Continued)

Attribute Name	Attribute ID	Attribute-Modifier	Description
PortSamplesResultExtended	0x001E	0x00000000	Extended Port Performance Data Sampling Results. See 16.1.4.10 PortSamplesResultExtended on page 963
SwPortVLCongestion	0x0030	0x00000000	Switch Port Congestion per VL. See 16.1.4.9 SwPortVLCongestion on page 962.

Table 227 Optional Performance Management Attribute / Method Map

Attribute Name	PerformanceGet()	PerformanceSet()
PortRcvErrorDetails	X	X
PortXmitDiscardDetails	X	X
PortOpRcvCounters	X	X
PortFlowCtlCounters	X	X
PortVLOpPackets	X	X
PortVLOpData	X	X
PortVLXmitFlowCtlUpdateErrors	X	X
PortVLXmitWaitCounters	X	X
PortSamplesResultExtended	X	X
PortCountersExtended	X	X
SwPortVLCongestion	X	X

16.1.4.1 PORTRCVERRORDETAILS

Table 228 PortRcvErrorDetails

Component	Access	Length (bits)	Offset (bits)	Description
Reserved	RO	8	0	Reserved
PortSelect	RW	8	8	Selects the port, as defined in Table 222 PortSamplesControl on page 934, for which the statistics are reported. Statistics are accumulated for all VLs on a port. If gathering data from all ports at once is supported (see Table 221 Performance Management ClassPortInfo:CapabilityMask on page 933), setting PortSelect to 0xFF will cause data from all valid ports to be accumulated. When selecting invalid port values, any results are undefined.

Table 228 PortRcvErrorDetails (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
CounterSelect	RW	16	16	When writing (Set), selects which counters are overwritten by the values specified in their respective fields. When reading (Get), this is ignored. Bit 0 - PortLocalPhysicalErrors Bit 1 - PortMalformedPacketErrors Bit 2 - PortBufferOverrunErrors Bit 3 - PortDLIDMappingErrors Bit 4 - PortVLMappingErrors Bit 5 - PortLoopingErrors Bits 6 to 15 - Reserved
PortLocalPhysicalErrors	RW	16	32	Total number of packets received on the port that contain local physical errors (ICRC, VCRC, FCCRC, and all physical errors that cause entry into the BAD PACKET or BAD PACKET DISCARD states of the packet receiver state machine).
PortMalformedPacketErrors	RW	16	48	Total number of packets received on the port that contain malformed packet errors • Data packets: LVer, length, VL • Link packets: operand, length, VL
PortBufferOverrunErrors	RW	16	64	Total number of packets received on the port that were discarded due to buffer overrun.
PortDLIDMappingErrors	RW	16	80	Total number of packets received on the port that were discarded because they could not be forwarded by the switch relay due to DLID mapping errors. DLID mapping errors occur when (a) DLID_Check=invalid as specified in Figure 52 Data Packet Check machine on page 176; or (b) for any of the discard reasons specified in C18-36:, C18-40:, C18-41: and C18-51:. This applies to switches only.
PortVLMappingErrors	RW	16	96	Packet discards due to VL mapping behavior are not considered errors, so the behavior of this counter is implementation-dependent. However, it is recommended that this counter be used to count the total number of packets received on the port that were discarded because they could not be forwarded by the switch relay due to VL mapping behavior 7.6.6 VL Mapping Within a Subnet on page 186
PortLoopingErrors	RW	16	112	Total number of packets received on the port that were discarded because they could not be forwarded by the switch relay due to looping errors (output port = input port). This applies to switches only.

16.1.4.2 PORTXMITDISCARDDETAILS

Table 229 PortXmitDiscardDetails

Component	Access	Length (bits)	Offset (bits)	Description
Reserved	RO	8	0	Reserved
PortSelect	RW	8	8	Selects the port, as defined in Table 222 PortSamplesControl on page 934, for which the statistics are reported. Statistics are accumulated for all VLs on a port. If gathering data from all ports at once is supported (see Table 221 Performance Management ClassPortInfo:CapabilityMask on page 933), setting PortSelect to 0xFF will cause data from all valid ports to be accumulated. When selecting invalid port values, any results are undefined.
CounterSelect	RW	16	16	When writing (Set), selects which counters are overwritten by the values specified in their respective fields. When reading (Get), this is ignored. Bit 0 - PortInactiveDiscards Bit 1 - PortNeighborMTUDiscards Bit 2 - PortSwLifetimeLimitDiscards Bit 3 - PortSwHOQLifetimeLimitDiscards Bits 4 to 15 - Reserved
PortInactiveDiscards	RW	16	32	Total number of outbound packets discarded by the port because it is not in the active state.
PortNeighborMTUDiscards	RW	16	48	Total number of outbound packets discarded by the port because packet length exceeded the PortInfo:NeighborMTU.
PortSwLifetimeLimitDiscards	RW	16	64	Total number of outbound packets discarded by the port because the Switch Lifetime Limit was exceeded. Applies to switches only.
PortSwHOQLifetimeLimitDiscards	RW	16	80	Total number of outbound packets discarded by the port because the switch HOQ Lifetime Limit was exceeded. Applies to switches only.

16.1.4.3 PORTOPRCVCOUNTERS

Table 230 PortOpRcvCounters

Component	Access	Length (bits)	Offset (bits)	Description
OpCode	RW	8	0	Selects the op code (as found in BTH) for which the statistics are reported. 0xFF means all op codes.

Table 230 PortOpRcvCounters (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
PortSelect	RW	8	8	Selects the port, as defined in Table 222 PortSamplesControl on page 934, for which the statistics are reported. Statistics are accumulated for all VLs on a port. If gathering data from all ports at once is supported (see Table 221 Performance Management ClassPortInfo:CapabilityMask on page 933), setting PortSelect to 0xFF will cause data from all valid ports to be accumulated. When selecting invalid port values, any results are undefined.
CounterSelect	RW	16	16	When writing (Set), selects which counters are overwritten by the values specified in their respective fields. When reading (Get), this is ignored. Bit 0 - PortOpRcvPkts Bit 1 - PortOpRcvData Bits 2 to 15 - Reserved
PortOpRcvPkts	RW	32	32	Total number of packets received without error on the port selected by PortSelect containing the opcode selected by OpCode.
PortOpRcvData	RW	32	64	Total number of data octets, divided by 4, received without error on all VLs from the port selected by PortSelect containing the opcode selected by OpCode. This includes all octets between (and not including) the start of packet delimiter and VCRC. It excludes all link packets. Implementers may choose to count data octets in groups larger than four but are encouraged to choose the smallest group possible. Results are still reported as a multiple of four octets.

16.1.4.4 PORTFLOWCTLCOUNTERS

Table 231 PortFlowCtlCounters

Component	Access	Length (bits)	Offset (bits)	Description
Reserved	RO	8	0	Reserved
PortSelect	RW	8	8	Selects the port, as defined in Table 222 PortSamplesControl on page 934, for which the statistics are reported. If gathering data from all ports at once is supported (see Table 221 Performance Management ClassPortInfo:CapabilityMask on page 933), setting PortSelect to 0xFF will cause data from all valid ports to be accumulated. When selecting invalid port values, any results are undefined.

Table 231 PortFlowCtlCounters (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
CounterSelect	RW	16	16	When writing (Set), selects which counters are overwritten by the values specified in their respective fields. When reading (Get), this is ignored. Bit 0 - PortXmitFlowPkts Bit 1 - PortRcvFlowPkts Bits 2 to 15 - Reserved
PortXmitFlowPkts	RW	32	32	Total number of flow control packets transmitted on the port selected by PortSelect
PortRcvFlowPkts	RW	32	64	Total number of flow control packets received on the port selected by PortSelect

16.1.4.5 PORTVLOPPACKETS

Table 232 PortVLOpPackets

Component	Access	Length (bits)	Offset (bits)	Description
OpCode	RW	8	0	Selects the op code (as found in BTH) for which the statistics are reported. 0xFF means all op codes.
PortSelect	RW	8	8	Selects the port, as defined in Table 222 PortSamplesControl on page 934, for which the statistics are reported. If gathering data from all ports at once is supported (see Table 221 Performance Management ClassPortInfo:CapabilityMask on page 933), setting PortSelect to 0xFF will cause data from all valid ports to be accumulated. When selecting invalid port values, any results are undefined.

Table 232 PortVLOpPackets (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
CounterSelect	RW	16	16	When writing (Set), selects which counters are overwritten by the values specified in their respective fields. When reading (Get), this is ignored. Bit 0 - PortVLOpPackets0 Bit 1 - PortVLOpPackets1 Bit 2 - PortVLOpPackets2 Bit 3 - PortVLOpPackets3 Bit 4 - PortVLOpPackets4 Bit 5 - PortVLOpPackets5 Bit 6 - PortVLOpPackets6 Bit 7 - PortVLOpPackets7 Bit 8 - PortVLOpPackets8 Bit 9 - PortVLOpPackets9 Bit 10 - PortVLOpPackets10 Bit 11 - PortVLOpPackets11 Bit 12 - PortVLOpPackets12 Bit 13 - PortVLOpPackets13 Bit 14 - PortVLOpPackets14 Bit 15 - PortVLOpPackets15
PortVLOpPackets0	RW	16	32	The total number of packets received without error on VL 0 of the port selected by PortSelect containing the opcode selected by OpCode
PortVLOpPackets1	RW	16	48	Similar count for VL1
PortVLOpPackets2	RW	16	64	Similar count for VL2
PortVLOpPackets3	RW	16	80	Similar count for VL3
PortVLOpPackets4	RW	16	96	Similar count for VL4
PortVLOpPackets5	RW	16	112	Similar count for VL5
PortVLOpPackets6	RW	16	128	Similar count for VL6
PortVLOpPackets7	RW	16	144	Similar count for VL7
PortVLOpPackets8	RW	16	160	Similar count for VL8
PortVLOpPackets9	RW	16	176	Similar count for VL9
PortVLOpPackets10	RW	16	192	Similar count for VL10
PortVLOpPackets11	RW	16	208	Similar count for VL11
PortVLOpPackets12	RW	16	224	Similar count for VL12
PortVLOpPackets13	RW	16	240	Similar count for VL13
PortVLOpPackets14	RW	16	256	Similar count for VL14

Table 232 PortVLOpPackets (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
PortVLOpPackets15	RW	16	272	Similar count for VL15

16.1.4.6 PORTVLOPDATA

Table 233 PortVLOpData

Component	Access	Length (bits)	Offset (bits)	Description
OpCode	RW	8	0	Selects the op code (as found in BTH) for which the statistics are reported. 0xFF means all op codes.
PortSelect	RW	8	8	Selects the port, as defined in Table 222 PortSamplesControl on page 934, for which the statistics are reported. If gathering data from all ports at once is supported (see Table 221 Performance Management ClassPortInfo:CapabilityMask on page 933), setting PortSelect to 0xFF will cause data from all valid ports to be accumulated. When selecting invalid port values, any results are undefined.
CounterSelect	RW	16	16	When writing (Set), selects which counters are overwritten by the values specified in their respective fields. When reading (Get), this is ignored. Bit 0 - PortVLOpData0 Bit 1 - PortVLOpData1 Bit 2 - PortVLOpData2 Bit 3 - PortVLOpData3 Bit 4 - PortVLOpData4 Bit 5 - PortVLOpData5 Bit 6 - PortVLOpData6 Bit 7 - PortVLOpData7 Bit 8 - PortVLOpData8 Bit 9 - PortVLOpData9 Bit 10 - PortVLOpData10 Bit 11 - PortVLOpData11 Bit 12 - PortVLOpData12 Bit 13 - PortVLOpData13 Bit 14 - PortVLOpData14 Bit 15 - PortVLOpData15

Table 233 PortVLOpData (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
PortVLOpData0	RW	32	32	Total number of data octets, divided by 4, received without error on VL 0 from the port selected by PortSelect containing the opcode selected by OpCode. This includes all octets between (and not including) the start of packet and VCRC. It excludes all link packets. Implementers may choose to count data octets in groups larger than four but are encouraged to choose the smallest group possible. Results are still reported as a multiple of four octets
PortVLOpData1	RW	32	64	Similar count for VL1
PortVLOpData2	RW	32	96	Similar count for VL2
PortVLOpData3	RW	32	128	Similar count for VL3
PortVLOpData4	RW	32	160	Similar count for VL4
PortVLOpData5	RW	32	192	Similar count for VL5
PortVLOpData6	RW	32	224	Similar count for VL6
PortVLOpData7	RW	32	256	Similar count for VL7
PortVLOpData8	RW	32	288	Similar count for VL8
PortVLOpData9	RW	32	320	Similar count for VL9
PortVLOpData10	RW	32	352	Similar count for VL10
PortVLOpData11	RW	32	384	Similar count for VL11
PortVLOpData12	RW	32	416	Similar count for VL12
PortVLOpData13	RW	32	448	Similar count for VL13
PortVLOpData14	RW	32	480	Similar count for VL14
PortVLOpData15	RW	32	512	Similar count for VL15

16.1.4.7 PORTVLXMITFLOWCTLUPDATEERRORS

Table 234 PortVLXmitFlowCtlUpdateErrors

Component	Access	Length (bits)	Offset (bits)	Description
Reserved	RO	8	0	Reserved

Table 234 PortVLXmitFlowCtlUpdateErrors (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
PortSelect	RW	8	8	Selects the port, as defined in Table 222 PortSamplesControl on page 934, for which the statistics are reported. If gathering data from all ports at once is supported (see Table 221 Performance Management ClassPortInfo:CapabilityMask on page 933), setting PortSelect to 0xFF will cause data from all valid ports to be accumulated. When selecting invalid port values, any results are undefined.
CounterSelect	RW	16	16	When writing (Set), selects which counters are overwritten by the values specified in their respective fields. When reading (Get), this is ignored. Bit 0 - PortVLXmitFlowCtlUpdateErrors0 Bit 1 - PortVLXmitFlowCtlUpdateErrors1 Bit 2 - PortVLXmitFlowCtlUpdateErrors2 Bit 3 - PortVLXmitFlowCtlUpdateErrors3 Bit 4 - PortVLXmitFlowCtlUpdateErrors4 Bit 5 - PortVLXmitFlowCtlUpdateErrors5 Bit 6 - PortVLXmitFlowCtlUpdateErrors6 Bit 7 - PortVLXmitFlowCtlUpdateErrors7 Bit 8 - PortVLXmitFlowCtlUpdateErrors8 Bit 9 - PortVLXmitFlowCtlUpdateErrors9 Bit 10 - PortVLXmitFlowCtlUpdateErrors10 Bit 11 - PortVLXmitFlowCtlUpdateErrors11 Bit 12 - PortVLXmitFlowCtlUpdateErrors12 Bit 13 - PortVLXmitFlowCtlUpdateErrors13 Bit 14 - PortVLXmitFlowCtlUpdateErrors14 Bit 15 - PortVLXmitFlowCtlUpdateErrors15
PortVLXmitFlowCtl UpdateErrors0	RW	2	32	Total number of flow control update errors on VL 0 on the port selected by PortSelect
PortVLXmitFlowCtl UpdateErrors1	RW	2	34	Similar count for VL1
PortVLXmitFlowCtl UpdateErrors2	RW	2	36	Similar count for VL2
PortVLXmitFlowCtl UpdateErrors3	RW	2	38	Similar count for VL3
PortVLXmitFlowCtl UpdateErrors4	RW	2	40	Similar count for VL4
PortVLXmitFlowCtl UpdateErrors5	RW	2	42	Similar count for VL5
PortVLXmitFlowCtl UpdateErrors6	RW	2	44	Similar count for VL6

Table 234 PortVLXmitFlowCtlUpdateErrors (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
PortVLXmitFlowCtl UpdateErrors7	RW	2	46	Similar count for VL7
PortVLXmitFlowCtl UpdateErrors8	RW	2	48	Similar count for VL8
PortVLXmitFlowCtl UpdateErrors9	RW	2	50	Similar count for VL9
PortVLXmitFlowCtl UpdateErrors10	RW	2	52	Similar count for VL10
PortVLXmitFlowCtl UpdateErrors11	RW	2	54	Similar count for VL11
PortVLXmitFlowCtl UpdateErrors12	RW	2	56	Similar count for VL12
PortVLXmitFlowCtl UpdateErrors13	RW	2	58	Similar count for VL13
PortVLXmitFlowCtl UpdateErrors14	RW	2	60	Similar count for VL14
PortVLXmitFlowCtl UpdateErrors15	RW	2	62	Similar count for VL15

16.1.4.8 PORTVLXMITWAITCOUNTERS

Table 235 PortVLXmitWaitCounters

Component	Access	Length (bits)	Offset (bits)	Description
Reserved	RO	8	0	Reserved
PortSelect	RW	8	8	Selects the port, as defined in Table 222 PortSamplesControl on page 934, for which the statistics are reported. If gathering data from all ports at once is supported (see Table 221 Performance Management ClassPortInfo:CapabilityMask on page 933), setting PortSelect to 0xFF will cause data from all valid ports to be accumulated. When selecting invalid port values, any results are undefined.

Table 235 PortVLXmitWaitCounters (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
CounterSelect	RW	16	16	When writing (Set), selects which counters are overwritten by the values specified in their respective fields. When reading (Get), this is ignored. Bit 0 - PortVLXmitWait0 Bit 1 - PortVLXmitWait1 Bit 2 - PortVLXmitWait2 Bit 3 - PortVLXmitWait3 Bit 4 - PortVLXmitWait4 Bit 5 - PortVLXmitWait5 Bit 6 - PortVLXmitWait6 Bit 7 - PortVLXmitWait7 Bit 8 - PortVLXmitWait8 Bit 9 - PortVLXmitWait9 Bit 10 - PortVLXmitWait10 Bit 11 - PortVLXmitWait11 Bit 12 - PortVLXmitWait12 Bit 13 - PortVLXmitWait13 Bit 14 - PortVLXmitWait14 Bit 15 - PortVLXmitWait15
PortVLXmitWait0	RW	16	32	Total number of ticks during which the port selected by PortSelect had data to transmit on VL0 but no data was sent during the entire tick either because of insufficient credits or because of lack of arbitration.
PortVLXmitWait1	RW	16	48	Similar count for VL1
PortVLXmitWait2	RW	16	64	Similar count for VL2
PortVLXmitWait3	RW	16	80	Similar count for VL3
PortVLXmitWait4	RW	16	96	Similar count for VL4
PortVLXmitWait5	RW	16	112	Similar count for VL5
PortVLXmitWait6	RW	16	128	Similar count for VL6
PortVLXmitWait7	RW	16	144	Similar count for VL7
PortVLXmitWait8	RW	16	160	Similar count for VL8
PortVLXmitWait9	RW	16	176	Similar count for VL9
PortVLXmitWait10	RW	16	192	Similar count for VL10
PortVLXmitWait11	RW	16	208	Similar count for VL11
PortVLXmitWait12	RW	16	224	Similar count for VL12
PortVLXmitWait13	RW	16	240	Similar count for VL13
PortVLXmitWait14	RW	16	256	Similar count for VL14

Table 235 PortVLXmitWaitCounters (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
PortVLXmitWait15	RW	16	272	Similar count for VL15

16.1.4.9 SWPORTVLCONGESTION

o16-2.1.2: SwPortVLCongestion **shall** not be supported by nodes other than switches.

Table 236 SwPortVLCongestion

Component	Access	Length (bits)	Offset (bits)	Description
Reserved	RO	8	0	Reserved
PortSelect	RW	8	8	Selects the port, as defined in Table 222 PortSamplesControl on page 934, for which the statistics are reported. If gathering data from all ports at once is supported (see Table 221 Performance Management ClassPortInfo:CapabilityMask on page 933), setting PortSelect to 0xFF will cause data from all valid ports to be accumulated. When selecting invalid port values, any results are undefined.
CounterSelect	RW	16	16	When writing (Set), selects which counters are overwritten by the values specified in their respective fields. When reading (Get), this is ignored. Bit 0 - SWPortVLCongestion0 Bit 1 - SWPortVLCongestion1 Bit 2 - SWPortVLCongestion2 Bit 3 - SWPortVLCongestion3 Bit 4 - SWPortVLCongestion4 Bit 5 - SWPortVLCongestion5 Bit 6 - SWPortVLCongestion6 Bit 7 - SWPortVLCongestion7 Bit 8 - SWPortVLCongestion8 Bit 9 - SWPortVLCongestion9 Bit 10 - SWPortVLCongestion10 Bit 11 - SWPortVLCongestion11 Bit 12 - SWPortVLCongestion12 Bit 13 - SWPortVLCongestion13 Bit 14 - SWPortVLCongestion14 Bit 15 - SWPortVLCongestion15
SWPortVLCongestion0	RW	16	32	Total number of packets to be transmitted on VL 0 of the output port selected by PortSelect that were discarded because of congestion. This includes the following reasons: • Switch Lifetime Limit exceeded • Switch HOQ Lifetime Limit exceeded

Table 236 SwPortVLCongestion (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
SWPortVLCongestion1	RW	16	48	Similar count for VL1.
SWPortVLCongestion2	RW	16	64	Similar count for VL2.
SWPortVLCongestion3	RW	16	80	Similar count for VL3.
SWPortVLCongestion4	RW	16	96	Similar count for VL4.
SWPortVLCongestion5	RW	16	112	Similar count for VL5.
SWPortVLCongestion6	RW	16	128	Similar count for VL6
SWPortVLCongestion7	RW	16	144	Similar count for VL7
SWPortVLCongestion8	RW	16	160	Similar count for VL8
SWPortVLCongestion9	RW	16	176	Similar count for VL9
SWPortVLCongestion10	RW	16	192	Similar count for VL10
SWPortVLCongestion11	RW	16	208	Similar count for VL11
SWPortVLCongestion12	RW	16	224	Similar count for VL12
SWPortVLCongestion13	RW	16	240	Similar count for VL13
SWPortVLCongestion14	RW	16	256	Similar count for VL14
SWPortVLCongestion15	RW	16	272	Similar count for VL15

16.1.4.10 PORTSAMPLESRESULTEXTENDED

Table 237 PortSamplesResultExtended

Component	Access	Length (bits)	Offset (bits)	Description
Tag	RO	16	0	Semantics identical to PortSamplesResult component Tag on page 944
Reserved	RO	14	16	Reserved
SampleStatus	RO	2	30	Semantics identical to PortSamplesResult component SampleStatus on page 944

Table 237 PortSamplesResultExtended (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
ExtendedWidth	RO	2	32	Indicates the actual width in bits of the PortSamplesResultExtended:Counter0 to 14, if supported. Ignored if IsExtendedWidthSupported capability bit is 0. • 00 = reserved • 01 = 48 bits • 10 = 64 bits • 11 = reserved Counters smaller than 64 bits shall be implemented as the least significant bits of the corresponding 64-bit attribute component, with the unimplemented upper bits of the component returning zeroes for Get( ) and ignored for Set( ).
Reserved	RO	32	32	Reserved
Counter0	RO	64	64	Semantics identical to PortSamplesResult component Counter0 on page 944
Counter1	RO	64	128	Semantics identical to PortSamplesResult component Counter1 on page 944
Counter2	RO	64	192	Semantics identical to PortSamplesResult component Counter2 on page 944
Counter3	RO	64	256	Semantics identical to PortSamplesResult component Counter3 on page 944
Counter4	RO	64	320	Semantics identical to PortSamplesResult component Counter4 on page 944
Counter5	RO	64	384	Semantics identical to PortSamplesResult component Counter5 on page 944
Counter6	RO	64	448	Semantics identical to PortSamplesResult component Counter6 on page 944
Counter7	RO	64	512	Semantics identical to PortSamplesResult component Counter7 on page 944
Counter8	RO	64	576	Semantics identical to PortSamplesResult component Counter8 on page 944
Counter9	RO	64	640	Semantics identical to PortSamplesResult component Counter9 on page 945
Counter10	RO	64	704	Semantics identical to PortSamplesResult component Counter10 on page 945
Counter11	RO	64	768	Semantics identical to PortSamplesResult component Counter11 on page 945
Counter12	RO	64	832	Semantics identical to PortSamplesResult component Counter12 on page 945

Table 237 PortSamplesResultExtended (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
Counter13	RO	64	896	Semantics identical to PortSamplesResult component Counter13 on page 945
Counter14	RO	64	960	Semantics identical to PortSamplesResult component Counter14 on page 945

16.1.4.11 PORTCOUNTERSEXTENDED

Table 238 PortSamplesResultExtended

Component	Access	Length (bits)	Offset (bits)	Description
Reserved	RO	8	0	Reserved
PortSelect	RW	8	8	Semantics identical to PortCounters component PortSelect on page 946
CounterSelect	RW	16	16	When writing (Set( )), selects which counters are affected by the operation. When reading (Get( )), this is ignored. • Bit 0 - PortXmitData • Bit 1 - PortRcvData • Bit 2 - PortXmitPkts • Bit 3 - PortRcvPkts • Bit 4 - PortUnicastXmitPkts • Bit 5 - PortUnicastRcvPkts • Bit 6 - PortMulticastXmitPkts • Bit 7 - PortMulticastRcvPkts • Bits 8-15 - Reserved
Reserved	RO	32	32	Reserved
PortXmitData	RW	64	64	Semantics identical to PortCounters component PortXmitData on page 948, except that if this attribute is supported, this component is not optional.
PortRcvData	RW	64	128	Semantics identical to PortCounters component PortRcvData on page 948, except that if this attribute is supported this component is not optional.
PortXmitPkts	RW	64	192	Semantics identical to PortCounters component PortXmitPkts on page 948, except that if this attribute is supported this component is not optional.
PortRcvPkts	RW	64	256	Semantics identical to PortCounters component PortRcvPkts on page 948, except that if this attribute is supported this component is not optional.

Table 238 PortSamplesResultExtended (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
PortUnicastXmitPkts	RW	64	320	Total number of unicast packets transmitted on all VLs from the port. This may include unicast packets with errors (see Figure 51 Packet Receiver State Machine on page 174 and Figure 52 Data Packet Check machine on page 176), and excludes link packets.
PortUnicastRcvPkts	RW	64	384	Total number of unicast packets, including unicast packets containing errors (see Figure 51 Packet Receiver State Machine on page 174 and Figure 52 Data Packet Check machine on page 176), and excluding link packets, received from all VLs on the port.
PortMultiCastXmitPkts	RW	64	448	Total number of multicast packets transmitted on all VLs from the port. This may include multicast packets with errors (see Figure 51 Packet Receiver State Machine on page 174 and Figure 52 Data Packet Check machine on page 176).
PortMultiCastRcvPkts	RW	64	512	Total number of multicast packets, including multicast packets containing errors (see Figure 51 Packet Receiver State Machine on page 174 and Figure 52 Data Packet Check machine on page 176) received from all VLs on the port.

16.1.5 PERFORMANCE MANAGEMENT STATUS

This section provides a consolidated interpretation of a large number of status-code-related compliance statements in Chapter 13: Management Model on page 709, Chapter 14: Subnet Management on page 794, and Chapter 16: General Services on page 930. It is provided here to only aid interoperability, and is not part of the specification.

In the following sections, an error status definitoin is specified. This definition provides a minimal requirement to report SMP (or PM GMP) errors. Note that adhering to this definition will satisfy all IB compliance rules. Any additional error conditions and error handling that are beyond what are specified in this section, without violating any IB compliance rules, are considered as additional efforts and therefore optional to the implementation.

16.1.5.1 MANDATORY PM ATTRIBUTE STATUS

In this implementation, there is an implicit precedence among various status settings. The higher the status value, the lower its precedence. Note that this status precedence approach is only one possible implementation. Other implementations may exist. This implementation does not violate nor override any of the defined compliance rules. Other implementations may return an error status without following this status

precedence. If multiple errors were detected, any error status bit may be returned in the GetResp( ).

Table 239 Mandatory PM Attribute Status

Precedence (0x1 is highest)	Status[4:2]	Violations
1	0x1	BaseVersion, ClassVersion, or their combinations is not supported by the vendor.
2	0x2	A request packet has been received with the method neither is a PerformanceGet() nor a PerformanceSet(). This packet may be discarded without a response or returned as status 2 with the “R” bit of the method field set to one without regarding the overall meaning of the method field or returned as status 2 with PerformanceGetResp().
3	0x3	It is not in any of the method/attribute combinations listed in Table 240 on page 967.
4	0x7	Containing at least one invalid component and/or AttributeModifier. See following sections for more details.
5	0x0	None of above violations were found.

Table 240 Valid Mandatory PM Method/Attribute Combinations

Attribute Name	Methods
ClassPortInfo	PerformanceGet()
PortSamplesControl	PerformanceGet() or PerformanceSet()
PortSamplesResult	PerformanceGet()
PortCounters	PerformanceGet() or PerformanceSet

16.1.5.1.1 PM ATTRIBUTEMODIFIER ERRORS (STATUS[4:2] = 0X7)

Table 241 PM AttributeModifier Errors

Attribute Name	AttributeModifier Violations
ClassPortInfo	AttributeModifier!= 0x0000_0000
PortSamplesControl	SampleMechanisms ≤ AttributeModifier ≤ 0xFFFF_FFFE
PortSamplesResult	SampleMechanisms ≤ AttributeModifier ≤ 0xFFFF_FFFE
PortCounters	AttributeModifier!= 0x0000_0000

16.1.5.1.2 PM ATTRIBUTE COMPONENT ERRORS (STATUS[4:2] = 0X7)

Table 242 PerformanceSet(ClassPortInfo) Component Errors

Violations
N/A

Table 243 PerformanceSet(PortSamplesControl) Component Errors

Attribute Component	Violations
OpCode[a]	Any of the reserved OpCode: • OpCode = 0x00010101 to 0x00011111 • OpCode = 0x00101100 to 0x00111111 • OpCode = 0x01010110 to 0x01011111 • OpCode = 0x01100000 to 0x01100011 • OpCode = 0x01100110 to 0x01111111 • OpCode = 0x10000000 to 0x10111111
	Any of the unsupported manufacturer specific OpCodes
PortSelect	PortSelect = 0x00[b]
	0xFF > PortSelect > NodeInfo:NumPorts
	PortSelect = 0xFF[c]

a. Only when sampling optional quantities that are OpCode specific.
b. Only if EnhancedPort0 = 0
c. Only if AllPortSelect = 0

Table 244 PerformanceSet(PortSamplesResult) Component Errors

Violations
N/A

Table 245 PerformanceSet(PortCounters) Component Errors

Attribute Component	Violations
PortSelect	PortSelect = 0x00[a]
	0xFF > PortSelect > NodeInfo:NumPorts
	PortSelect = 0xFF[b]

a. Only if EnhancedPort0 = 0.
b. Only if AllPortSelect = 0.

16.1.5.2 OPTIONAL PM ATTRIBUTE STATUS

In this implementation, there is an implicit precedence among various status settings. The higher the status value, the lower its precedence. This status precedence implementation is only one possible implementation. It does not violate nor override any of the defined compliance rules. Other implementations may return an error status without following this status precedence. If multiple errors were detected, any error status bit may be returned in the GetResp( ).

Table 246 Optional PM Attribute Status

Precedence (0x1 is highest)	Status[4:2]	Violations
1	0x1	BaseVersion, ClassVersion, or their combinations is not supported by the vendor.
2	0x2	Not PerformanceGet() or PerformanceSet(). Base on the current specification, this error cannot be reported and the packet will be discarded.
3	0x3	It is not in any of the method/attribute combinations listed in Table 247 on page 969.
4	0x7	Containing at least one invalid component and/or AttributeModifier. See following sections for more details.
5	0x0	None of above violations were found.

Table 247 Valid Optional PM Method/Attribute Combinations

Attribute Name	Methods
PortRcvErrorDetails	PerformanceGet() or PerformanceSet()
PortXmitDiscardDetails	PerformanceGet() or PerformanceSet()
PortOpRcvCounters	PerformanceGet() or PerformanceSet()
PortFlowCtlCounters	PerformanceGet() or PerformanceSet()
PortVLOpPackets	PerformanceGet() or PerformanceSet()
PortVLOpData	PerformanceGet() or PerformanceSet()

Table 247 Valid Optional PM Method/Attribute Combinations

Attribute Name	Methods
PortVLXmitFlowCtlUpdateErrors	PerformanceGet() or PerformanceSet()
PortVLXmitWaitCounters	PerformanceGet() or PerformanceSet()
SwPortVLCongestion	PerformanceGet() or PerformanceSet()

16.1.5.2.1 PM ATTRIBUTEMODIFIER ERRORS (STATUS[4:2] = 0X7)

Table 248 PM AttributeModifier Errors

Attribute Name	AttributeModifier Violations
PortRcvErrorDetails	AttributeModifier!= 0x0000_0000
PortXmitDiscardDetails	AttributeModifier!= 0x0000_0000
PortOpRcvCounters	AttributeModifier!= 0x0000_0000
PortFlowCtlCounters	AttributeModifier!= 0x0000_0000
PortVLOpPackets	AttributeModifier!= 0x0000_0000
PortVLOpData	AttributeModifier!= 0x0000_0000
PortVLXmitFlowCtlUpdateErrors	AttributeModifier!= 0x0000_0000
PortVLXmitWaitCounters	AttributeModifier!= 0x0000_0000
SwPortVLCongestion	AttributeModifier!= 0x0000_0000

16.1.5.2.2 PM ATTRIBUTE COMPONENT ERRORS (STATUS[4:2] = 0X7)

Table 249 PerformanceSet(PortRcvErrorDetails) Component Errors

Attribute Component	Violations
PortSelect	PortSelect = 0x00[a]
	0xFF > PortSelect > NodeInfo:NumPorts
	PortSelect = 0xFF[b]
CounterSelect	Any bit in CounterSelect[15:6] is set to 1

a. Only if EnhancedPort0 = 0.
b. Only if AllPortSelect = 0.

Table 250 PerformanceSet(PortXmitDiscardDetails) Component Errors

Attribute Component	Violations
PortSelect	PortSelect = 0x00[a]
	0xFF > PortSelect > NodeInfo:NumPorts
	PortSelect = 0xFF[b]
CounterSelect	Any bit in CounterSelect[15:4] is set to 1

a. Only if EnhancedPort0 = 0.
b. Only if AllPortSelect = 0.

Table 251 PerformanceSet(PortOpRcvCounters) Component Errors

Attribute Component	Violations
OpCode	Any of the reserved OpCode: • OpCode = 0x00010101 to 0x00011111 • OpCode = 0x00101100 to 0x00111111 • OpCode = 0x01010110 to 0x01011111 • OpCode = 0x01100000 to 0x01100011 • OpCode = 0x01100110 to 0x01111111 • OpCode = 0x10000000 to 0x10111111
	Any of the unsupported manufacturer specific OpCodes
PortSelect	PortSelect = 0x00[a]
	0xFF > PortSelect > NodeInfo:NumPorts
	PortSelect = 0xFF[b]
CounterSelect	Any bit in CounterSelect[15:2] is set to 1

a. Only if EnhancedPort0 = 0.
b. Only if AllPortSelect = 0.

Table 252 PerformanceSet(PortFlowCtlCounters) Component Errors

Attribute Component	Violations
PortSelect	PortSelect = 0x00[a]
	0xFF > PortSelect > NodeInfo:NumPorts
	PortSelect = 0xFF[b]
CounterSelect	Any bit in CounterSelect[15:2] is set to 1

a. Only if EnhancedPort0 = 0.
b. Only if AllPortSelect = 0.

Table 253 PerformanceSet(PortVLOpPackets) Component Errors

Attribute Component	Violations
OpCode	Any of the reserved OpCode: • OpCode = 0x00010101 to 0x00011111 • OpCode = 0x00101100 to 0x00111111 • OpCode = 0x01010110 to 0x01011111 • OpCode = 0x01100000 to 0x01100011 • OpCode = 0x01100110 to 0x01111111 • OpCode = 0x10000000 to 0x10111111
	Any of the unsupported manufacturer specific OpCodes
PortSelect	PortSelect = 0x00[a]
	0xFF > PortSelect > NodeInfo:NumPorts
	PortSelect = 0xFF[b]

a. Only if EnhancedPort0 = 0.
b. Only if AllPortSelect = 0.

Table 254 PerformanceSet(PortVLOpData) Component Errors

Attribute Component	Violations
OpCode	Any of the reserved OpCode: • OpCode = 0x00010101 to 0x00011111 • OpCode = 0x00101100 to 0x00111111 • OpCode = 0x01010110 to 0x01011111 • OpCode = 0x01100000 to 0x01100011 • OpCode = 0x01100110 to 0x01111111 • OpCode = 0x10000000 to 0x10111111
	Any of the unsupported manufacturer specific OpCodes
PortSelect	PortSelect = 0x00[a]
	0xFF > PortSelect > NodeInfo:NumPorts
	PortSelect = 0xFF[b]

a. Only if EnhancedPort0 = 0.
b. Only if AllPortSelect = 0.

Table 255 PerformanceSet(PortVLXmitFlowCtlUpdateErrors) Component Errors

Attribute Component	Violations
PortSelect	PortSelect = 0x00[a]
	0xFF > PortSelect > NodeInfo:NumPorts
	PortSelect = 0xFF[b]

a. Only if EnhancedPort0 = 0.

b. Only if AllPortSelect = 0.

Table 256 PerformanceSet(PortVLXmitWaitCounters) Component Errors

Attribute Component	Violations
PortSelect	PortSelect = 0x00[a]
	0xFF > PortSelect > NodeInfo:NumPorts
	PortSelect = 0xFF[b]

a. Only if EnhancedPort0 = 0.
b. Only if AllPortSelect = 0.

Table 257 PerformanceSet(PortVLXmitFlowCtlUpdateErrors) Component Errors

Attribute Component	Violations
PortSelect	PortSelect = 0x00[a]
	0xFF > PortSelect > NodeInfo:NumPorts
	PortSelect = 0xFF[b]

a. Only if EnhancedPort0 = 0.
b. Only if AllPortSelect = 0.

16.2 BASEBOARD MANAGEMENT

C16-9: All nodes **shall** have a Baseboard Management Agent.

C16-9.1.1: The Baseboard Manager (BM) **shall** register its services with the SA via SubnAdmSet(ServiceRecords) using the ServiceName “BaseboardManager.IBTA”. See 15.2.5.14 ServiceRecord on page 895 and Service Names in the Annex “Application Specific Identifiers.”

This section describes the Management Datagrams used to transport Baseboard Management commands across the fabric. For more information regarding Hardware Management of IB Modules, non-Modules (IB devices whose packaging are different from an IB Module form factor), and Chassis, see InfiniBand Architecture Specification, Volume 2, Chapter “Hardware Management.” A simplified overview is presented here.

The SM and SMA are shown in Figure 199 Baseboard Management Architecture on page 974 strictly to indicate that Baseboard Management and Subnet Management are independent, separate entities in the fabric providing non-overlapping functionality.

The Baseboard Manager is a software entity that manages the hardware via Baseboard Management messages. From the BM, these messages

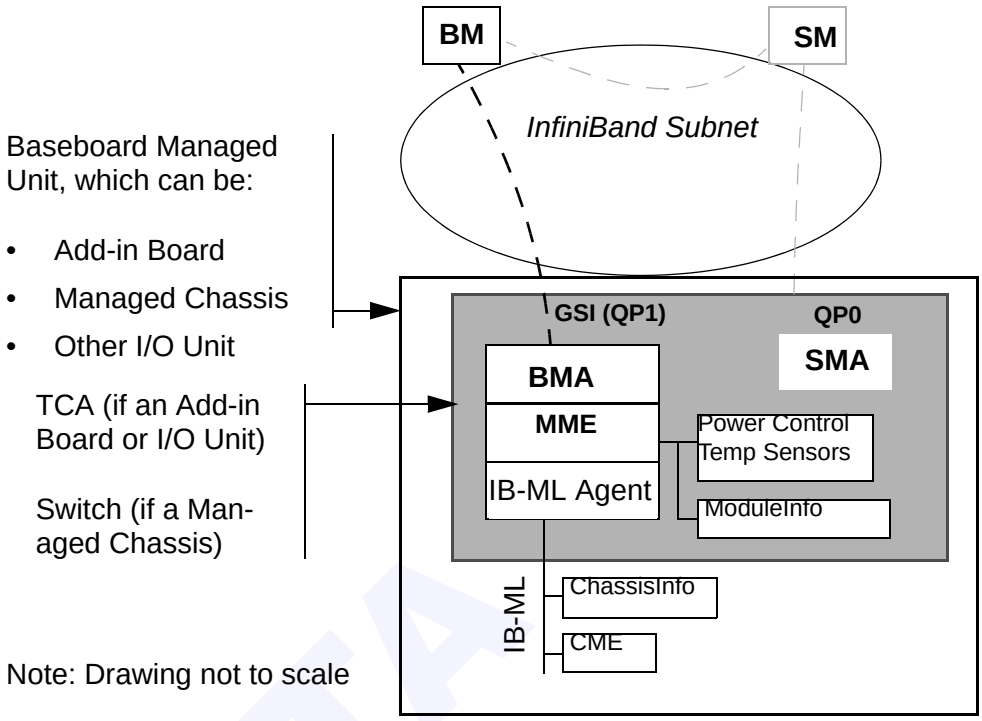


Figure 199 Baseboard Management Architecture

are tunneled (encapsulated in MADs) through the IBA fabric to the Baseboard Management Agent (BMA), which then recognizes the message and forwards it to the Module Management Entity (MME). The MME processes the embedded Baseboard Management commands. In some cases, this results in the MME generating corresponding messages and transactions on a present InfiniBand Management Link (IB-ML). IB-ML messages may interface with a present Chassis Management Entity (CME).

The BM may use Subnet Administration to retrieve information regarding the nodes discovered in the IBA subnet, such as addresses, capabilities, types.

Subnet Administration provides basic discovery information that is common to all IBA endnodes, regardless of the type of Endnode in the subnet as described above. It does not provide information beyond this, such as VPD, chassis management data, and any other information under Baseboard Management control. This information is “discovered” through baseboard management.

A Baseboard Managed Unit can be either an IB-Module as defined in Volume 2 of the InfiniBand Architecture Specification, a form factor other than what is defined in Volume 2 (a non-Module), or a Managed Chassis.

Protocol-aware IB-Modules handle the send/receive of the Baseboard Management MADs. The MADs are addressed using the LID of any end-port of the IB device on the Module. A Managed Chassis may contain a switch which handles the send/receive of the Baseboard Management MADs. The MADs are addressed using the LID of the switch.

16.2.1 MAD FORMAT

C16-10: The datagrams in the Baseboard management class **shall** conform to the MAD format and use as specified in 13.4 Management Datagrams on page 717 and further customized in Figure 200 Baseboard Management MAD Format on page 975 and Table 258 Baseboard Management MAD Fields on page 975 below.

bytes	bits 31-24	bits 23-16	bits 15-8	bits 7-0
0	Common MAD Header			
...				
20				
24	B_Key			
28				
32	Reserved			
...				
60				
64	Data			
...				
252				

Figure 200 Baseboard Management MAD Format

Table 258 Baseboard Management MAD Fields

Field Name	Length	Description
Common MAD Header	24 bytes	Common MAD as described in 13.4.2 Management Datagram Format on page 718
B_Key	8 bytes	BM specific key. See 16.2.4 B_Key General Use on page 982 for definition and use.
Reserved	32 bytes	Reserved

Table 258 Baseboard Management MAD Fields (Continued)

Field Name	Length	Description
Baseboard Management Data	192 bytes	Attribute data as defined in the InfiniBand Architecture Specification, Volume 2, Management Command Section, which is refined by the Attribute ID and the AttributeModifier. See Table 261 Baseboard Management Attributes on page 978 for valid Attribute ID and Modifier values.

16.2.1.1 STATUS FIELD

The Status field is described in 13.4.7 Status Field on page 731. No class-specific bits are defined.

Table 259 Baseboard Management Status Field

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.7 Status Field on page 731
8-15	-	Class-specific bits are reserved

16.2.2 METHODS

The Baseboard Management class uses a subset of the common methods described in 13.4.5 Management Class Methods on page 721.

C16-10.1.1: A Baseboard Management Agent **shall** support the methods listed in Table 260 Baseboard Management Methods on page 976. All method type values not listed in the Table are reserved.

Table 260 Baseboard Management Methods

Method Type	Value	Description
BMGet()	0x01	Request a get (read) of an attribute
BMSet()	0x02	Request a set (write) of an attribute
BMGetResp()	0x81	Response from a Get() or Set() request
BMSend()	0x03	Send Baseboard Management attribute (this can be the attribute for an encapsulated Baseboard Management request [command] or response.)
BMTrap()	0x05	Notify an event occurred
BMTrapRepress()	0x07	Block repetition of notification
BMReport()	0x06	Forward an event previously subscribed for
BMReportResp()	0x86	Reply to a BMReport() method

Figure 201 BM Initiated IB-ML Command on page 977 is an example of a transaction. Its semantics are described in the InfiniBand Architecture

Specification, Volume 2, Chapter “Hardware Management,” Section “Management Commands.”

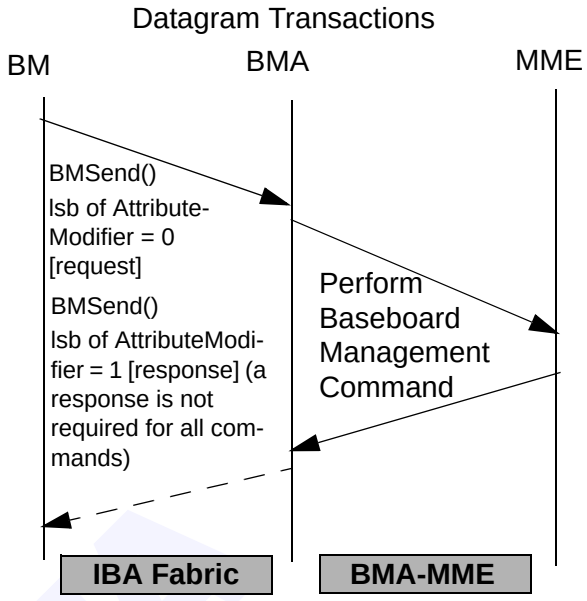


Figure 201 BM Initiated IB-ML Command

Requests and response capabilities are symmetric; i.e., BMSend() may be used to send a request from the MME as well as a response. Similarly, the Baseboard Manager may use BMSend() to deliver a response. Figure 202 IB-ML Initiated Command on page 978 illustrates the path that would be taken if a CME generated a request to the Baseboard Manager by delivering the request via a module’s IB-ML -to- IB functionality.

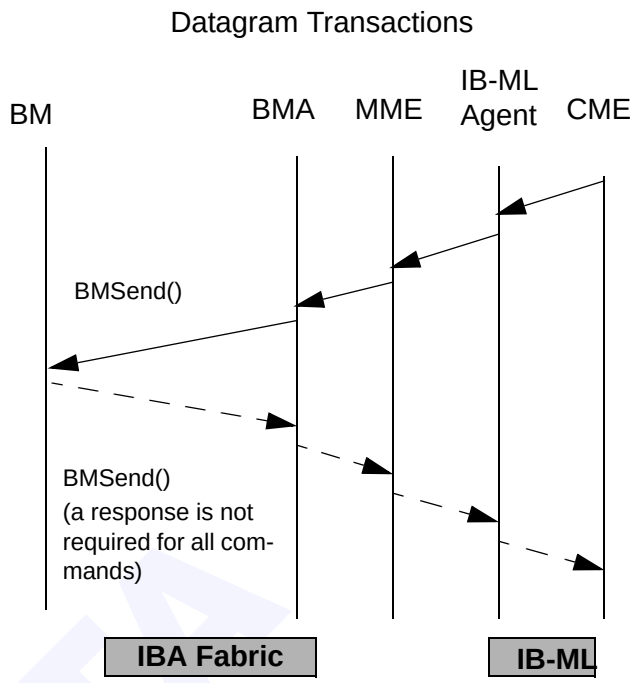


Figure 202 IB-ML Initiated Command

16.2.3 ATTRIBUTES

C16-10.1.2: A Baseboard Management Agent **shall** support the attributes listed in Table 261 Baseboard Management Attributes on page 978, Table 262 Baseboard Management Attribute / Method Map on page 979 and the InfiniBand Architecture Specification, Volume 2, Chapter "Hardware Management." All attribute IDs not listed in Table 261: Baseboard Management Attributes are reserved.

Table 261 Baseboard Management Attributes

Attribute Name	Attribute ID	AttributeModifier[a]	Description
ClassPortInfo	0x0001	0x00000000	General and port-specific information for the BM class. See 16.2.3.1 ClassPortInfo on page 980.
Notice	0x0002	0x00000000	Information regarding a Trap. See 16.2.3.2 Notice on page 980.
BKeyInfo	0x0010	0x00000000	B_Key information for the node. See 16.2.3.3 BKeyInfo on page 982.

Table 261 Baseboard Management Attributes (Continued)

Attribute Name	Attribute ID	AttributeModifier[a]	Description
WriteVPD	0x0020	0x00000000 / 0x00000001	See the InfiniBand Architecture Specification, Volume 2, Chapter "Hardware Management" for all these attributes.
ReadVPD	0x0021	0x00000000 / 0x00000001	
ResetIBML	0x0022	0x00000000 / 0x00000001	
SetModulePMControl	0x0023	0x00000000 / 0x00000001	
GetModulePMControl	0x0024	0x00000000 / 0x00000001	
SetUnitPMControl	0x0025	0x00000000 / 0x00000001	
GetUnitPMControl	0x0026	0x00000000 / 0x00000001	
SetIOCPMControl	0x0027	0x00000000 / 0x00000001	
GetIOCPMControl	0x0028	0x00000000 / 0x00000001	
SetModuleState	0x0029	0x00000000 / 0x00000001	
SetModuleAttention	0x002A	0x00000000 / 0x00000001	
GetModuleStatus	0x002B	0x00000000 / 0x00000001	
IB2IBML	0x002C	0x00000000 / 0x00000001	
IB2CME	0x002D	0x00000000 / 0x00000001	
IB2MME	0x002E	0x00000000 / 0x00000001	
OEM	0x002F	0x00000000 / 0x00000001	

a. Where two AttributeModifiers are listed, the least significant bit of the AttributeModifier is used to differentiate BMSend() requests from responses. Refer to InfiniBand Architecture Specification, Volume 2, Chapter "Hardware Management" for more details.

Table 262 Baseboard Management Attribute / Method Map

Attribute Name	BMGet()	BMSet()	BMSend()	BMTrap()
ClassPortInfo	X	X		
Notice	X	X		X
BKeyInfo	X	X		
WriteVPD			X	
ReadVPD			X	
ResetIBML			X	
SetModulePMControl			X	
GetModulePMControl			X	
SetUnitPMControl			X	

Table 262 Baseboard Management Attribute / Method Map

Attribute Name	BMGet()	BMSet()	BMSend()	BMTrap()
GetUnitPMControl			X	
SetIOCPMControl			X	
GetIOCPMControl			X	
SetModuleState			X	
SetModuleAttention			X	
GetModuleStatus			X	
IB2IBML			X	
IB2CME			X	
IB2MME			X	
OEM			X	

16.2.3.1 CLASSPORTINFO

The ClassPortInfo attribute is described in 13.4.8.1 ClassPortInfo on page 734. Class-specific bits of the Baseboard Management ClassPortInfo:CapabilityMask are defined in Table 263 Baseboard Management ClassPortInfo:CapabilityMask on page 980.

Table 263 Baseboard Management ClassPortInfo:CapabilityMask

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.8.1 ClassPortInfo on page 734
8	IsIBMLSupported	Direct Access to IB-ML is supported
9	IsBKeyNVRAM	B_Key is in NVRAM
10-15		Reserved

16.2.3.2 NOTICE

The Notice attribute is described in 13.4.8.2 Notice on page 737.

Table 264 Baseboard Management Traps

Name	Type	Number	DataDetails
BKeyViolation	Security	259	Bad B_Key, <B_Key> from <LIDADDR>/<GIDADDR>/<QP> attempted <METHOD> with <ATTRIBUTEID> and <ATTRIBUTEMODIFIER>
BMTraps	Informational	260	<BMTrapDataLength> <BMTrapType> <BMTrapTypeModifier> <BMTrapData>

o16-3: The BKeyViolation trap uses the following layout for the DataDetails component of the Notice attribute, see Table 265 Notice DataDetails For Trap 259 on page 981. Fields **shall** be filled with the information corresponding to the description of a given trap.

Table 265 Notice DataDetails For Trap 259

Field	Length(bits)	Description
LIDADDR	16	Local Identifier
METHOD	8	Method
Reserved	8	Reserved
ATTRIBUTEID	16	Attribute ID
ATTRIBUTE MODIFIER	32	AttributeModifier
Reserved	8	Reserved
QP	24	Queue Pair
BKEY	64	B_Key
GIDADDR	128	Global Identifier. If no GRH is present in the offending packet, this field **shall** be filled with zeroes.
Padding	128	**Shall** be ignored on read. Content is unspecified.

o16-3.1.1: The BMTraps use the layout shown in Table 266 Notice DataDetails For Trap 260 on page 981 for the DataDetails component of the Notice attribute. Fields **shall** be filled with the information corresponding to the description of a given trap.

Table 266 Notice DataDetails For Trap 260

Field	Length(bits)	Description
BMTrapDataLength	8	Number of TrapData bytes to included
BMTrapType	8	Indicates reason for trap 0x00= Generic MME 0x01 = OEM MME 0x02 = CME_RTR 0x03 = WRE 0x04 = Generic CME 0x05 = OEM CME All others - reserved

Table 266 Notice DataDetails For Trap 260 (Continued)

Field	Length(bits)	Description
BMTrapTypeModifier	24	Varies based on TrapType: • holds 3-byte OEM ID if Trap Type is OEM MME or OEM CME • holds SlotSelector, 0x00, 0x00 if TrapType is CME_RTR • holds 0x00, 0x00, 0x00 (reserved) if TrapType is Generic MME, WRE, or Generic CME
BMTrapData	8*TrapDataLength	Data bytes for TrapType of OEM MME or OEM CME.
Padding	392-(8*TrapDataLength)	**Shall** be ignored on read. Content is unspecified.

16.2.3.3 BKEYINFO

Table 267 BKeyInfo

Component	Access	Length (bits)	Offset (bits)	Description
B_Key	RW	64	0	The 8-byte Baseboard Management key used in all BM MADs by all valid BMs. A value of 0 means no B_Key check is ever done by the BMA.
B_KeyProtectBit	RW	1	64	See 16.2.4.3 B_Key Operation on page 984 for details.
Reserved	RW	15	65	Reserved
B_KeyLeasePeriod	RW	16	80	Timer value used to indicate how long the B_Key ProtectBit is to remain non zero after a BMSet(BKeyInfo) MAD that failed a B_Key check is dropped. The value of the timer indicates the number of seconds for the lease period. With a 16 bit counter, the period can range from one second to approximately 18 hours. 0 **shall** mean infinite. See 16.2.4.5 B_Key Recovery on page 984 for details.
B_KeyViolations	RO	16	96	Number of MADs that have been received at this node since power-on or reset that have been dropped due to a failed B_Key check if such a counter is implemented. Otherwise this **shall** be 0xFFFF.

16.2.3.4 IB-ML ATTRIBUTES

See *InfiniBand Architecture Specification, Volume 2, Chapter "Hardware Management", Section "Management Commands"* for a description of the BM class specific attributes and their format.

16.2.4 B_KEY GENERAL USE

The BM includes the Baseboard Management Key (B_Key) in the BM MAD to obtain authorization. The B_Key is used to authenticate a trusted source. This model assumes that the fabric has some level of physical security.

16.2.4.1 B_KEY ASSUMPTIONS

1) To use the correct key for each node, the BM or a higher-level B_Key manager keeps track of the keys for the nodes that it is managing.
2) If a backup BM exists, it shares the B_Keys for ease of fail-over.
3) A BM may have exclusive access to a set of nodes, by using a B_Key which is only known by that BM and those particular nodes. Since nodes reply to Baseboard Manager's requests using their own B_Key, if a Baseboard Manager assigns more than one B_Key to devices on its management domain, it needs to run on an HCA whose BMA can respond to all such B_Keys.
4) The BM sets the B_Key, B_KeyProtectBit, and B_KeyLeasePeriod in the BKeyInfo Attribute with one BMSet(BkeyInfo) MAD. A successful completion of this assignment indicates to the BM that it has taken ownership of the node.

16.2.4.2 B_KEY PROTECTION SCOPE

Each BMA in a node has one B_Key. Table 268 B_Key Protection Scope on page 983 shows the scope protected by that B_Key. The semantics are explicitly defined in InfiniBand Architecture Specification, Volume 2, Chapter "Hardware Management."

Table 268 B_Key Protection Scope

Source	Targeted Entity	Protection
BM	Read and Writes to • ClassPortInfo (e.g., BM LID in TrapLID) • BKeyInfo (e.g., B_Key, B_KeyProtectBit)	yes
BM	Attributes causing reads from and writes to IB-ML • ModuleInfo[a] • IB-module Specific Data • ChassisInfo[b] • CME[b] • Other IB-ML devices	yes
IB-ML Managed Unit	Attributes causing reads from and writes to IB-ML • IB-module VPD • IB-module Specific Data • ChassisInfo[b] • Other IB-ML devices	no

a. The IB-Module vendor protects the factory-programmed portion of ModuleInfo against writes even if a proper B_Key is provided.
b. The Chassis vendor protects the factory-programmed portion of ChassisInfo against writes even if a proper B_Key is provided. If further protection is desired, the CME or the Chassis provides it.

16.2.4.3 B_KEY OPERATION

C16-11: The BMA **shall** check the B_Key contained in incoming MADs.

The success and effect of the check depends on the value of the BKeyInfo:B_Key and BKeyInfo:B_KeyProtectBit of the BMA and on the method and attribute contained in the incoming MAD.

Table 269 B_Key Check

BMA's B_Key	BMA's B_Key Protection Bit	MAD's method	Success
zero	any	any	yes
non-zero	any	BMSet(), BMSend()	if MAD's B_Key equals BMA's B_Key
non-zero	0	BMGet()	yes
non-zero	1	BMGet()	yes[a]

a. Even though the check succeeds, the B_Key value in the BKeyInfo attribute **shall** be returned as zero.

C16-12: If B_Key check fails, the BMA **shall**:

1) Drop the MAD.
2) Increment a B_Key Violation counter if supported.
3) Send a BKeyViolation trap if traps are supported by the BMA.
4) Start a countdown timer with the B_Key lease period value.

16.2.4.4 B_KEY INITIALIZATION

C16-13: At power up or reset, the BKeyInfo:B_Key, BKeyInfo:B_KeyProtectBit and BKeyInfo:B_KeyLeasePeriod **shall** be set to zero if NVRAM is not used; otherwise, they **shall** be set to the values stored in NVRAM.

If the B_Key-related components are not stored in NVRAM, the BKeyInfo:B_Key, BKeyInfo:B_KeyProtectBit and BKeyInfo:B_KeyLeasePeriod components may be set by the BM. Initialization of BKeyInfo:B_KeyLeasePeriod to a value of zero notwithstanding, whenever a pot's B_Key-related components are not stored in NVRAM, a BM may use BMSet(BKeyInfo) to assign the subsequent BKeyInfo:B_Key, BKeyInfo:B_KeyProtectBit and BKeyInfo:B_KeyLeasePeriod.

16.2.4.5 B_KEY RECOVERY

The B_Key lease period timer starts when a B_Key check fails. At this time, the node sends a trap to the BM (if traps are supported and if the BM stored its information in the trap components of the ClassPortInfo attribute). This trap serves as a request to the BM to refresh the lease period

by issuing a BMSet(BKeyInfo). A successful BMSet(BKeyInfo) will stop the timer and will rearm it.

If the BM that originally set the B_Key has gone away, then the lease period expires—clearing the BKeyInfo:B_KeyProtectBit and allowing anyone to read (and then set) the BkeyInfo:B_Key.

In the case where a node starts with NVRAM, BKeyInfo:B_Key and BKeyInfo:B_KeyProtectBit set and the TrapLID is zero (because no BM has come around to set it), the node has no BM to send the trap to. In this case, the node does not send the trap and the lease period timer will expire, causing eventual take over by a new BM.

With the BMGet(BKeyInfo), any BM can detect whether a BKeyInfo:B_Key is set (although hidden) based on the BKeyInfo:B_KeyProtectBit. If the BKeyInfo:B_KeyProtectBit is set, the BKeyInfo:B_Key is set and hidden. Otherwise the returned BKeyInfo:B_Key is the real one even if it is zero.

16.2.4.6 LEVELS OF PROTECTION

There are four different protection levels based on the B_Key, depending on the system requirements.

Table 270 Protection Levels

B_Key	B_KeyProtectBit	B_KeyLeasePeriod	Description
0	any	any	No protection provided. Any BM can issue sets and sends.
non-zero	0	n/a	Protection provided, but allows BMs to read the BKeyInfo:B_Key in the node.
non-zero	1	non-zero	Protection provided and does not allow anyone to read the B_Key in the node until the lease period has expired. The B_Key lease period is a mechanism to allow the B_Key to be protected only for a given amount of time.
non-zero	1	0	Protection provided and does not allow the B_Key in the node to be read by other BMs. It must be noted that if the lease period was set to 0 (infinite) and the BM that set it dies, there is no possibility for other BMs to ever read it. So if the B_Key is not provided by some unspecified way to the other BMs, the BMA of this node will never be accessible again.

16.3 DEVICE MANAGEMENT

The Device Management Agent is optional.

o16-3.1.2: If there is a Device Manager, it **shall** register its services with the SA via SubnAdmSet(ServiceRecord) using the ServiceName “Device-Manager.IBTA.” See 15.2.5.14 ServiceRecord on page 895 and Service Names in the Annex “Application Specific Identifiers.”

IO Devices and I/O controllers (IOC) are not directly connected to the IBA fabric. An I/O Unit (IOU) containing one or more IOCs is attached to the fabric via a TCA. The TCA is responsible for receiving packets from the fabric and delivering complete, valid messages to IOCs, and vice-versa. The TCA might use memory resources supplied by the IOC to assemble the packet and notify the IOC when the complete packet is available for consumption. IOC is then responsible for executing I/O requests such as network sends and receives or disk reads and writes over a device specific interface such as Ethernet or SCSI.

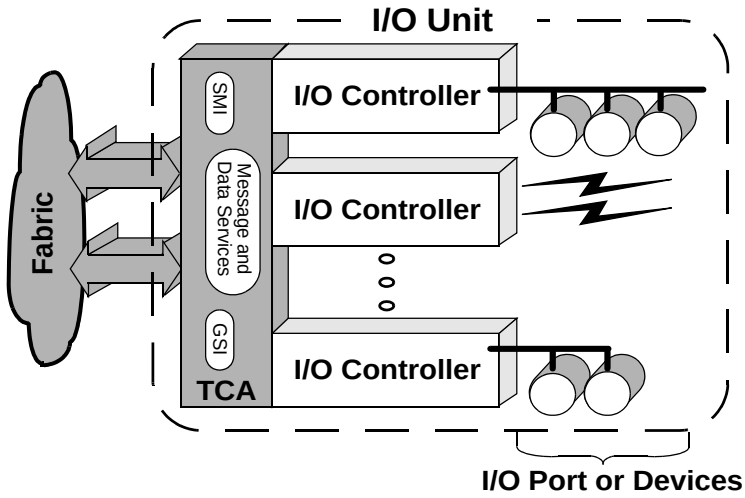
This section does not address direct management of end devices such as disk drives but focuses on the infrastructure, related methods, data formats and attributes to support IOU/IOC management over the fabric. This section defines mechanisms to send and receive device management packets between two fabric attachment points such as a HCA and a TCA. The mechanisms required to translate MADs into a format that the end devices understand and how the data is delivered and retrieved from an end device is device-specific and therefore is not addressed in this specification.

The IBA is based on message passing. For IOU and IOC, the messages fall into three classes: fabric configuration, unit management/configuration, and I/O transaction:

- Fabric configuration messages that are processed by the Subnet Management Agent (SMA) are defined in Chapter 14: Subnet Management on page 794.
- Messages specific to configuring and managing a device that are received through the General Services Interface (GSI) are described in this section.
- IO transaction messages are not defined in this document. I/O transaction messages include those messages used by an initiator to request I/O services from an IOC, messages containing user or application data, and messages used by the IOC to provide a completion notification (ending status) to the requester. Also included in this class are in-band configuration messages (parameters, etc.) directed only to an IOC, and not to the larger IOU as a whole. These messages travel as I/O requests but perform management functions specific to the I/O controller.

Although this chapter tends to use language implying that an IOU “contains” IOCs, there are no restrictions on how IOCs are connected to, or served by, the TCA. Figure 203 Architectural Model for an I/O Unit on

page 987 provides the architectural and connection models for an IOU, consisting of a TCA and one or more IOCs.



ure 203 Architectural Model for an I/O Unit

16.3.1 MAD FORMAT

o16-4: The datagrams in the Device Management class **shall** conform to the MAD format and use as specified in 13.4 Management Datagrams on page 717 and further customized in Figure 204 Device Management MAD Format on page 987 and Table 271 Device Management MAD Fields on page 987 below.

bytes	bits 31-24	bits 23-16	bits 15-8	bits 7-0
0	Common MAD Header			
...				
20				
24	Reserved			
...				
60				
64	Data			
...				
252				

Figure 204 Device Management MAD Format

Table 271 Device Management MAD Fields

Field	Length	Description
Common MAD Header	24 bytes	Common MAD Header as described in 13.4.2 Management Datagram Format on page 718

Table 271 Device Management MAD Fields (Continued)

Field	Length	Description
Reserved	40 bytes	Reserved
DevMgt Data	192 bytes	192 bytes of Device Management payload. The structure and content depends upon the Method, Attribute and AttributeModifier fields in the header.

16.3.1.1 STATUS FIELD

The Status field is described in 13.4.7 Status Field on page 731. Some class-specific bits are defined.

Table 272 Device Management Status Field

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.7 Status Field on page 731
8	NoResponse	IOC Not responding
9	NoServiceEntries	Service Entries are not supported
10-14	-	Reserved
15	GeneralFailure	IOC General Failure

16.3.2 METHODS

Among the services that a TCA provides to an initiating client is a mechanism to deliver detailed information about the I/O resources (e.g., IOCs) supported by the IOU. This information transcends a simple count of the number of IOCs supported to provide details of each IOC such as a GUID, a vendor-unique ID, product revision levels, and other information that is specific to a given IOC. The purpose of the detailed information is to let a system configuration manager allocate the IOU's resources to various clients located on the IBA fabric, and to provide a common way for host resource managers to determine the characteristics of IOUs and IOCs. This allows the proper driver to be associated with each controller.

The profiles are requested and returned through the GSI, which is an unreliable datagram service. The actual access QP and DLID may be redirected by the GSI. The IOUnitInfo attribute contains information on the number of IOCs the unit can support (IOUnitInfo:MaxControllers). This value is the length of the IOUnitInfo:ControllerList, which has an entry for every possible controller "slot" (which may be physical or logical). Each entry in the ControllerList component shows whether a controller is present. For each controller, the IOControllerProfile attribute contains information such as the type of controller and the identity of the IOC's vendor. Each controller has a ServiceEntries attribute associated with it. ServiceEntries is a table of ServiceIDs that the controller advertises to its

clients. The format of the IOUnitInfo, IOControllerProfile and ServiceEntries structures are defined in 16.3.3 Attributes on page 989.

o16-4.1.1: If the Device Management Class is supported, the Device Management Agent **shall** support the methods listed in Table 273: Device Management Methods. All method type values not listed in the Table are reserved.

Table 273 Device Management Methods

Method Type	Value	Description
DevMgtGet()	0x01	Request an IOU to return (read) Device Management class attributes such as profile or a list of controllers currently installed.
DevMgtSet()	0x02	Request an IOU to set (write) an attribute. The object will issue a DevMgtGetResp() as a response.
DevMgtGetResp()	0x81	IOU responds to an attribute Get or Set request.
DevMgtTrap()	0x05	Unsolicited datagram sent to the Device Management entity. Contains the Notice Attribute as defined in 13.4.8.2 Notice on page 737 to identify the trap.
DevMgtTrapRepress()	0x07	Block repetition of notification.
DevMgtReport()	0x06	Forward an event previously subscribed for
DevMgtReportResp()	0x86	Reply to a DevMgtReport() method

16.3.3 ATTRIBUTES

This section specifies the format of the attributes used for managing the IOU. Messages used as part of I/O transactions are not specified in this document. The term “device” refers to actual devices sitting behind IOCs. The way they are numbered is implementation-specific

o16-4.1.2: If the Device Management class is implemented, the Device Management Agent **shall** support the attributes listed in Table 274 Device Management Attributes on page 989 and Table 275 Device Management Attribute / Method Map on page 991. All attribute IDs not listed in Table 274: Device Management Attributes are reserved.

Table 274 Device Management Attributes

Attribute Name	Attribute ID	Attribute-Modifier[a]	Description
ClassPortInfo	0x0001	0x0000_0000	See 16.3.3.1 ClassPortInfo on page 991.
Notice	0x0002	0x0000_0000	See 16.3.3.2 Notice on page 992.
IOUnitInfo	0x0010	0x0000_0000	List of all IOCs present in a given IOU. Each IOU may support up to 0xFF controllers. See 16.3.3.3 IOUnitInfo on page 992.

Table 274 Device Management Attributes (Continued)

Attribute Name	Attribute ID	Attribute-Modifier[a]	Description
IOControllerProfile	0x0011	0x0000_0001-0x0000_00FF	IOC Profile Information. AttributeModifier identifies the IOC. See 16.3.3.4 IOControllerProfile on page 993.
ServiceEntries	0x0012	0x0001_0000-0x00FF_FFFF	List of supported services and their associated Service IDs. See 16.3.3.5 ServiceEntries on page 995. Each IOC has a table with at most 255 ServiceEntries. The AttributeModifier is structured as follows: • the upper 16 bits identify the IOC • the lower 16 bits specify a range of up to four Service Entries to be retrieved. The lower 8 bits specify the beginning of the range and the upper 8 bits specify the end of the range.
Reserved	0x0013-0x001F	0x0000_0000-0xFFFF_FFFF	Reserved
DiagnosticTimeout	0x0020	0x0000_0000 - 0xFFFF_FFFF	Response indicates maximum time for completion of diagnostic test. Target device is identified by the AttributeModifier. Tests not completing within this period may indicate device failure. Specified in multiples of milliseconds. See 16.3.3.6 DiagnosticTimeout on page 996.
PrepareToTest	0x0021	0x0000_0000 - 0xFFFF_FFFF	A Set with this Attribute instructs the device specified by the AttributeModifier to prepare for diagnostic test. A Get of this Attribute will result in the appropriate Response Status being set as follows: 0x0000 = Ready for diagnostic test 0x0100 = Invalid AttributeModifier 0x0200 = Device not ready 0x0400 = Device not responding 0x0800 = Diagnostics not supported 0x1000 - 0x8000 = Reserved See 16.3.3.7 PrepareToTest on page 996.
TestDeviceOnce	0x0022	0x0000_0000 - 0xFFFF_FFFF	A Set instructs the device specified by the AttributeModifier to initiate a single diagnostic test and run it once. Vendor-unique attributes (AttributeID values 0xFF00 - 0xFFFF) may be defined to initiate specific test instructions. See 16.3.3.8 TestDeviceOnce on page 996.
TestDeviceLoop	0x0023	0x0000_0000 - 0xFFFF_FFFF	A Set instructs the device specified by the AttributeModifier to initiate a single diagnostic test and run it continuously in a loop. Vendor-unique attributes (AttributeID values 0xFF00 - 0xFFFF) may be defined to initiate specific test instructions. See 16.3.3.9 TestDeviceLoop on page 996.
DiagCode	0x0024	0x0000_0000 - 0xFFFF_FFFF	Vendor-specific diagnostic information for the device specified by the AttributeModifier. See 16.3.3.10 DiagCode on page 996.
Reserved	0x0025-0xFEFF	0x0000_0000 - 0xFFFF_FFFF	Reserved

Table 274 Device Management Attributes (Continued)

Attribute Name	Attribute ID	Attribute-Modifier[a]	Description
Vendor specific	0xFF00-0xFFFF	0x0000_0000 - 0xFFFF_FFFF	Vendor-unique attribute values may be defined to deliver specific test instructions.

a. The AttributeModifier for a diagnostic attribute specifies the I/O component. An AttributeModifier of zero specifies the I/O unit itself. If IOUnitInfo:DiagDeviceID is one, the least significant 8 bits of a non-zero AttributeModifier designates the IOC. Other bits of the AttributeModifier (except the msb) are implementation specific and can be used to identify additional I/O components such as secondary I/O ports. The recommended practice is that if the additional bits are zero, then the diagnostic attribute applies to the IOC itself and non-zero values specify secondary I/O components.

Table 275 Device Management Attribute / Method Map

Attribute Name	DevMgtGet()	DevMgtSet()	DevMgtTrap()
ClassPortInfo	X	X	
Notice	X	X	X
IOUnitInfo	X		
IOControllerProfile	X		
ServiceEntries	X		
DiagnosticTimeout	X		
PrepareToTest	X	X	
TestDeviceOnce		X	
TestDeviceLoop		X	
DiagCode	X		

16.3.3.1 CLASSPORTINFO

The ClassPortInfo attribute is described in 13.4.8.1 ClassPortInfo on page 734. No class-specific bits are defined.

Table 276 Device Management ClassPortInfo:CapabilityMask

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.8.1 ClassPortInfo on page 734
8-15	-	Class-specific bits are reserved

16.3.3.2 NOTICE

The Notice attribute is described in 13.4.8.2 Notice on page 737. It is used for one optional generic trap.

Table 277 Device Management Traps

Name	Type	Number	DataDetails
ReadyToTest	Informational	514	Device <DEVICE> readiness is <STATUS>, where status is the same as would have been returned by a Get(PrepareToTest) with device as the AttributeModifier.

o16-5: Device Management Traps use the following layout for the DataDetails component of the Notice attribute, see Table 278 Notice DataDetails For Trap 514 on page 992. Fields **shall** be filled with the information corresponding to the description of a given trap.

Table 278 Notice DataDetails For Trap 514

Field	Length(bits)	Description
STATUS	16	Readiness status
DEVICE	32	Device number
Padding	384	**Shall** be ignored on read. Content is unspecified.

16.3.3.3 IOUNITINFO

Table 279 IOUnitInfo

Component	Access	Length (bits)	Offset (bits)	Description
Change_ID	RO	16	0	Incremented, with rollover, by any change to ControllerList.
Max Controllers	RO	8	16	Number of slots in ControllerList.
Reserved	RO	6	24	Reserved
DiagDeviceID	RO	1	30	A zero indicates that the AttributeModifier in diagnostic attributes is I/O unit vendor-specific. A one indicates that the AttributeModifier's least significant 8 bits identify the IOC Slot Number. This when this bit is one, diagnostics are per IOC. When this bit is zero, the I/O unit vendor defines how the AttributeModifier is interpreted. This allows diagnostics to be executed on objects behind the IOC.
Option ROM	RO	1	31	Indicates presence of Option ROM. 1 = Present; 0 = Absent.

Table 279 IOUnitInfo (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
ControllerList	RO	1024	32	A series of 4-bit nibbles with each representing a slot in the IOU. Each 4-bit nibble can take the following values: • 0x0 = IOC not installed • 0x1 = IOC present • 0x2-0xE = reserved • 0xF = slot does not exist Bits 7-4 of the first byte (lowest offset) represent slot 1, bits 3-0 represent slot 2, bits 7-4 of the second byte represent slot 3, bits 3-0 represent slot 4, and so on.

16.3.3.4 IOCONTROLLERPROFILE

Table 280 IOControllerProfile

Component	Access	Length (bits)	Offset (bits)	Description
GUID	RO	64	0	An EUI-64 GUID used to uniquely identify the controller. This could be the same one as the Node/Port GUID if there is only one controller.
VendorID	RO	24	64	IO controller vendor ID, IEEE format
Reserved	RO	8	88	Reserved for alignment
IocDeviceID	RO	32	96	A number assigned by the vendor to identify the type of controller. This can be used by an Operating System to select a device driver.
Device Version	RO	16	128	A number assigned by the vendor to identify the device version.
Reserved	RO	16	144	Reserved for alignment.
Subsystem VendorID	RO	24	160	ID of the vendor of the enclosure, if any, in which the I/O controller resides in IEEE format; otherwise zero.
Reserved	RO	8	184	Reserved for alignment.
SubsystemID	RO	32	192	A number identifying the subsystem where the controller resides.
IOClass	RO	16	224	0x0000-0xFFFE = Reserved for I/O classes encompassed by the InfiniBand Architecture. Refer to Annex “Application-Specific Identifiers.” 0xFFFF = Vendor-specific.

Table 280 IOControllerProfile (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
IOSubclass	RO	16	240	0x0000-0xFFFE = Reserved for I/O subclasses encompassed by the InfiniBand Architecture. Refer to Annex “Application-Specific Identifiers.” 0xFFFF = Vendor-specific. This shall be set to 0xFFFF if the I/O Class component is set to 0xFFFF.
Protocol	RO	16	256	0x0000-0xFFFE = Reserved for I/O protocols encompassed by the InfiniBand Architecture. Refer to Annex “Application-Specific Identifiers.” 0xFFFF = Vendor-specific. This shall be set to 0xFFFF if the I/O Class component is set to 0xFFFF.
Protocol Version	RO	16	272	Protocol specific.
Reserved	RO	16	288	Reserved
Reserved	RO	16	304	Reserved
Send Message Queue Depth	RO	16	320	Maximum depth of the Send Message Queue.
Reserved	RO	8	336	Reserved for alignment
RDMA Read Queue Depth	RO	8	344	Maximum depth of the per-channel RDMA Read Queue.
Send Message Size	RO	32	352	Maximum size of Send Messages in bytes.
RDMA Transfer Size	RO	32	384	Maximum size of outbound RDMA transfers initiated by the IOC - in bytes.
Controller Operations Capability Mask	RO	8	416	Supported operation types of this I/O controller. A bit set to 1 for affirmation of supported capability. Bit: Name; Description 0: ST; Send Messages To IOCs 1: SF; Send Messages From IOCs 2: RT; RDMA Read Requests To IOCs 3: RF; RDMA Read Requests From IOCs 4: WT; RDMA Write Requests To IOCs 5: WF; RDMA Write Requests From IOCs 6: AT; Atomic Operations To IOCs 7: AF; Atomic Operations From IOCs
Reserved	RO	8	424	Reserved
Service Entries	RO	8	432	Number of entries in the ServiceEntries table.
Reserved	RO	72	440	Reserved
ID String	RO	512	512	UTF-8 encoded string for identifying the controller to operator.

An I/O Controller represents a QP consumer on the target side that provides a particular I/O function via an I/O protocol. The IOControllerProfile attribute provides information that enables a host to identify the I/O function and load the appropriate I/O driver. The IOC function can be matched with a proprietary driver by its vendor information components or with a generic driver by its Class, Subclass, and protocol components (Refer to Annex A1 “I/O Annex” for driver matching rules).

Each IOC lists one or more ServiceEntries, and each ServiceEntry identifies an instance of I/O service that the IOC provides. Examples of why an IOC might have multiple services entries are (1) when the IOC supports multiple protocols; (2) when the IOC uses a different QP for each set of I/O resources it provides; and (3) when the IOC supports different service criteria. For example, an IOC that supports SCSI RDMA Protocol (SRP) would provide a service entry for each SCSI target port and if the IOC supported another protocol (such as a proprietary I/O protocol) it would also provide additional ServiceEntries for that protocol. Additionally, if that IOC supports one or more management protocols (such as for a JBOD or RAID configuration program) it might also have additional ServiceEntries for them.

16.3.3.5 SERVICEENTRIES

Table 281 ServiceEntries

Component	Access	Length (bits)	Offset (bits)	Description
ServiceName_1	RO	320	0	UTF-8 encoded, null-terminated name of the service.
ServiceID_1	RO	64	320	An identifier of the associated Service.
ServiceName_2	RO	320	384	UTF-8 encoded, null-terminated name of the service.
ServiceID_2	RO	64	704	An identifier of the associated Service.
ServiceName_3	RO	320	768	UTF-8 encoded, null-terminated name of the service.
ServiceID_3	RO	64	1088	An identifier of the associated Service.
ServiceName_4	RO	320	1152	UTF-8 encoded, null-terminated name of the service.
ServiceID_4	RO	64	1472	An identifier of the associated Service.

16.3.3.6 DIAGNOSTICTIMEOUT

Table 282 DiagnosticTimeout

Component	Access	Length (bits)	Offset (bits)	Description
MaxDiagTime	RO	32	0	Maximum time to finish a diagnostic operation in milliseconds

16.3.3.7 PREPARETOTEST

Table 283 PrepareToTest

Component	Access	Length(bits)	Offset (bits)	Description
-	-	-		This attribute does not have any components

16.3.3.8 TESTDEVICEONCE

Table 284 TestDeviceOnce

Component	Access	Length(bits)	Offset (bits)	Description
-	-	-		This attribute does not have any components

16.3.3.9 TESTDEVICELOOP

Table 285 TestDeviceLoop

Component	Access	Length(bits)	Offset (bits)	Description
-	-	-		This attribute does not have any components

16.3.3.10 DIAGCODE

Table 286 DiagCode

Component	Access	Length(bits)	Offset (bits)	Description
DiagCode	RO	16	0	16-bit diagnostic code. A value of 0x0000 means “device operational”; all other values are IOU vendor-specific.
Vendor-specific				

16.3.4 DEVICE DIAGNOSTIC FRAMEWORK

Device Diagnostics allows the identification of faults in devices behind the target channel adapter. As such, it complements other sections of this specification that describe how problems at the fabric and node level may be identified and isolated.

The device diagnostic framework is intended to support tests within an active fabric. It is versatile enough, however, to accommodate vendor-unique approaches that may include retrieval of power-on data. It should be noted that some, and perhaps most, devices may not permit simultaneous use of I/O transaction messages and diagnostics. Unless data is flushed from internal buffers, for example, corruption or loss of user data might occur. Further, it is expected that the diagnostics tests would require setting the device to an initial, known state. For that reason, provision will be made to put the device into a “ready” state prior to test, which will likely cause I/O transactions to be held off. This may, in turn, cause established connections to time out, and other management notices to be sent.

In general, device diagnostics should be used with great care, and with full understanding of the potential impact to I/O transactions to the target device. It is best used during periods of initial configuration, major maintenance, or as a tool of last resort.

16.3.4.1 BEHAVIORS

The Device Management class of MADs (see 16.3 Device Management on page 985) is used for diagnostics. Within that class, standard methods as defined in Table 273 Device Management Methods on page 989 are utilized. Attributes specific to device diagnostics are defined by which vendor-supplied tests may be invoked, and the results of completed tests then determined. The AttributeModifier indicates the object under test. If IOUnitInfo:DiagDeviceID is one, then a non-zero AttributeModifier (ignoring the MSB) indicates the IOC and a zero AttributeModifier (ignoring the MSB) indicates the entire IOU. Results are reported in the DIAGCODE attribute. The first 16 bits provide an overall status of the device under test; a value of zero indicates that the device is operational, and all other values are vendor-specific. The attribute data following the 16-bit DIAGCODE is vendor-specific.

The PrepareToTest attribute within the DevMgtSet() method places the device into a test-ready state. The time required to complete this step is not predictable, as it may involve flushing data from cache memory, reinitializing SCSI ports, etc. The device indicates its readiness for test by signaling the IOU to send an Informational Trap.

Alternatively, a DevMgtGet() method on this attribute will return information pertaining to the specific device's ability or readiness for test. This al-

lows the status to be polled on a periodic basis, or to determine that the device does not support diagnostic tests.

Two modes are provided for initiating diagnostics: single test mode and continuous test mode. In single test mode, a single test sequence is initiated by setting an appropriate AttributeModifier for the TestDeviceOnce attribute, with MSB=0. The DevMgt Agent rejects a DevMgtSet(TestDeviceOnce) with AttributeModifier MSB=1. Once initiated, this vendor-defined test will run to completion. Because tests will vary by device technology and by vendor, the time-to-completion is inherently unpredictable. To detect errant devices which are unable to complete their diagnostic test, a DiagnosticTimeout attribute may be retrieved in advance of test initiation, which indicates the maximum allowable period for completion. Results of the completed diagnostic test are obtained through the DiagCode attribute of the DevMgtGetResp() method.

The continuous test mode can assist in detecting problems that are transient in nature, be used to initiate endurance-related tests. The continuous-test mode is initiated by setting an appropriate AttributeModifier for the TestDeviceLoop attribute, with MSB=1. Results of the last completed diagnostic test are obtained through the DiagCode attribute of the DevMgtGetResp() method.

The DevMgt Agent aborts the continuous test mode when it receives a DevMgtSet(TestDeviceLoop) with AttributeModifier MSB=0. In this case the DiagCode is indeterminate. The DevMgt Agent terminates the continuous test mode after it completes the current pass when it receives a DevMgtSet(TestDeviceOnce) with AttributeModifier MSB=0. In this case the DiagCode is valid.

Interpretation of results obtained through the DevMgtGetResp() method is vendor-specific.

It is beyond the scope of this specification to define a set of white-box, technology-specific diagnostic tests. Rather, the intent is to allow initiation of a vendor-supplied test sequence, for which the expected outcome would be either success or failure. The DiagCode format, however, allows flexibility for the vendor to provide specific, coded information about the test results.

16.4 SNMP TUNNELING

The SNMP Tunneling Agent is optional.

This section describes the Management Datagrams used to report native SNMP tunneling over the IBA.

SNMP, or Simple Network Management Protocol, consists of a set of standards for network management, a protocol, and a database specification to uniformly address managed information objects.

SNMP tunneling is a supported option to the InfiniBand architecture as a Management Datagram service. Devices advertise support for the SNMP tunneling service by use of the IsSNMPSupported Capability in PortInfo Attribute. If the value is non-zero a given device may be queried via the GSI for the QP and LID to access the SNMP service. Note that this capability allows for another port to supply SNMP tunneling services by proxy.

This section describes the required class-dependent behavior of the datagrams in this class.

16.4.1 MAD FORMAT

o16-6: The datagrams in the SNMP tunneling class **shall** conform to the MAD format and use as specified in 13.4 Management Datagrams on page 717 and further customized in Figure 205 SNMP Tunneling MAD Format on page 999 and Table 287 SNMP Tunneling MAD Fields on page 999 below.

bytes	bits 31-24	bits 23-16	bits 15-8	bits 7-0
0	Common MAD Header			
...				
20				
24	Reserved			
...				
52				
56	Raddress			
60	Payload Length	Segment Number	Source LID	
64	Data			
...				
252				

Figure 205 SNMP Tunneling MAD Format

Table 287 SNMP Tunneling MAD Fields

Field Name	Length	Description
Common MAD Header	24 bytes	Common MAD Header as described in 13.4.2 Management Datagram Format on page 718

Table 287 SNMP Tunneling MAD Fields (Continued)

Field Name	Length	Description
Reserved	32 bytes	Reserved
Raddress	4 bytes	Opaque address field that is used by SNMP agent to forward SNMP packets using SNMP redirect features.
Payload Length	1 byte	Number of valid data bytes in the SNMP segment being transferred.
Segment Number	1 byte	Segment number of a segmented SNMP packet.
Source LID	2 bytes	Local address of the SNMP packet sender.
Data	192 bytes	Attribute data is mapped bit for bit from the format described in the following sections to the start of this data field. If the attribute is smaller than the data field, the content of the remainder of the data field is unspecified.

16.4.1.1 STATUS FIELD

The Status field is described in 13.4.7 Status Field on page 731. No class-specific bits are defined.

Table 288 SNMP Tunneling Status Field

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.7 Status Field on page 731
8-15	-	Reserved (class-specific bits)

16.4.2 METHODS

This class utilizes the common methods described in 13.4.5 Management Class Methods on page 721

o16-6.1.1: If the SNMP Tunneling Management Class is supported, the SNMP Tunneling Agent **shall** support the methods listed in Table 289: SNMP Tunneling Methods. All method type values not listed in the Table are reserved.

Table 289 SNMP Tunneling Methods

Method Type	Value	Description
SnmpGet()	0x01	Request a get (read) of an Attribute.
SnmpSet()	0x02	Request a set (write) of an Attribute.
SnmpGetResp()	0x81	Response from a Get() or Set() request.
SnmpSend()	0x03	Send an Attribute to a node.

16.4.3 ATTRIBUTES

o16-6.1.2: If the SNMP Tunneling Management class is implemented, the SNMP Tunneling Management Agent **shall** support the attributes listed in Table 290 SNMP Tunneling Attributes on page 1001 and Table 291 SNMP Tunneling Attribute / Method Map on page 1001. All attribute IDs not listed in Table 290: SNMP Tunneling Attributes are reserved.

Table 290 SNMP Tunneling Attributes

Attribute Name	Attribute ID	Attribute Modifier	Description	
ClassPortInfo	0x0001	0x0000_0000	See 16.4.3.1 ClassPortInfo on page 1001.	
(none)	0x0010	any	Use of this attribute is vendor-specific	
PduInfo	0x0011	0x0000_0001	First SNMP segment	See 16.4.3.3 PduInfo on page 1002
		0x0000_0002	Intermediate SNMP segment	
		0x0000_0003	Last SNMP segment	
		0x0000_0004	First and Last SNMP segment	
		0x8000_0001	First redirected SNMP segment	
		0x8000_0002	Intermediate redirected SNMP segment	
		0x8000_0003	Last redirected SNMP segment	
		0x8000_0004	First and Last redirected SNMP segment	

Table 291 SNMP Tunneling Attribute / Method Map

Attribute Name	SnmpGet()	SnmpSet()	SnmpSend()
ClassPortInfo	X		
CommunityInfo			X
PduInfo			X

16.4.3.1 CLASSPORTINFO

The ClassPortInfo attribute is described in 13.4.8.1 ClassPortInfo on page 734. No class-specific bits are defined.

Table 292 SNMP Tunneling ClassPortInfo:CapabilityMask

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.8.1 ClassPortInfo on page 734
8-15	-	Reserved (class-specific bits)

16.4.3.2 OBSOLETE SECTION

This section is obsolete and has been deleted

16.4.3.3 PDUINFO

Table 293 PduInfo

Component	Settability	Length (bits)	Offset (bits)	Description
PduData	RO	1536	0	Data Segment of an SNMP message

16.4.4 OPERATIONS

This Figure is obsolete and has been deleted

Figure 206 This Figure is Obsolete and Has Been Deleted

A packet consists of one or more segments. If necessary, a packet will be segmented at the source, transmitted, and reassembled at the target. Using the SNMP datagram, the source specifies the SnmpSend method, PduInfo Attribute, then sets the AttributeModifier, segment number (if the packet is segmented), and the payload length fields to delimit and account for segments.

o16-7: When SNMP packets **shall** be segmented into multiple MADs, the data field of all but the last MAD transferred **shall** be completely filled (192 bytes of data).

o16-8: If any segment of a multiMAD transfer is not received within the timeout as specified in 13.4.6.3 Timeout/Timer Usage on page 730, then that entire MAD **shall** be discarded.

o16-9: The Transaction ID field of all the MADs of the SNMP packet **shall** be the same, and **shall** conform to the uniqueness of Transaction IDs as described in 13.4.6.4 TransactionID usage on page 731.

Because the destination is already known by the sender of the MAD packet, there is no need to include it in the MAD packet. However, because the sender may be expecting a response from the agent receiving this SNMP request, the original source LID is provided so the agent knows where to send a reply.

The SNMP management class can transfer a packet of up to 49,152 bytes. This is enough to accommodate any incoming SNMP/UDP packet and allow for flexibility if management packets arrive from a transport other than TCP/UDP.

o16-10: When the pieces are reassembled, the SNMP Message **shall** be extracted and passed up to the agent or manager for processing.

If a MAD is marked First and Last with the AttributeModifier, it is the only segment in the packet. No segmentation has occurred so no reassembly is required, and the SNMP packet is extracted and passed up to the next layer.

If SNMP Redirect is specified in the AttributeModifier, the packet is meant for a target managed by the proxy agent processing the packet. The proxy agent will need to parse the packet to extract the Raddress value of the final destination to reformat the PDU for further transport along the new interconnect.

16.4.4.1 SNMP TARGETS FOR BEYOND THE INFINIBAND ENDNODE

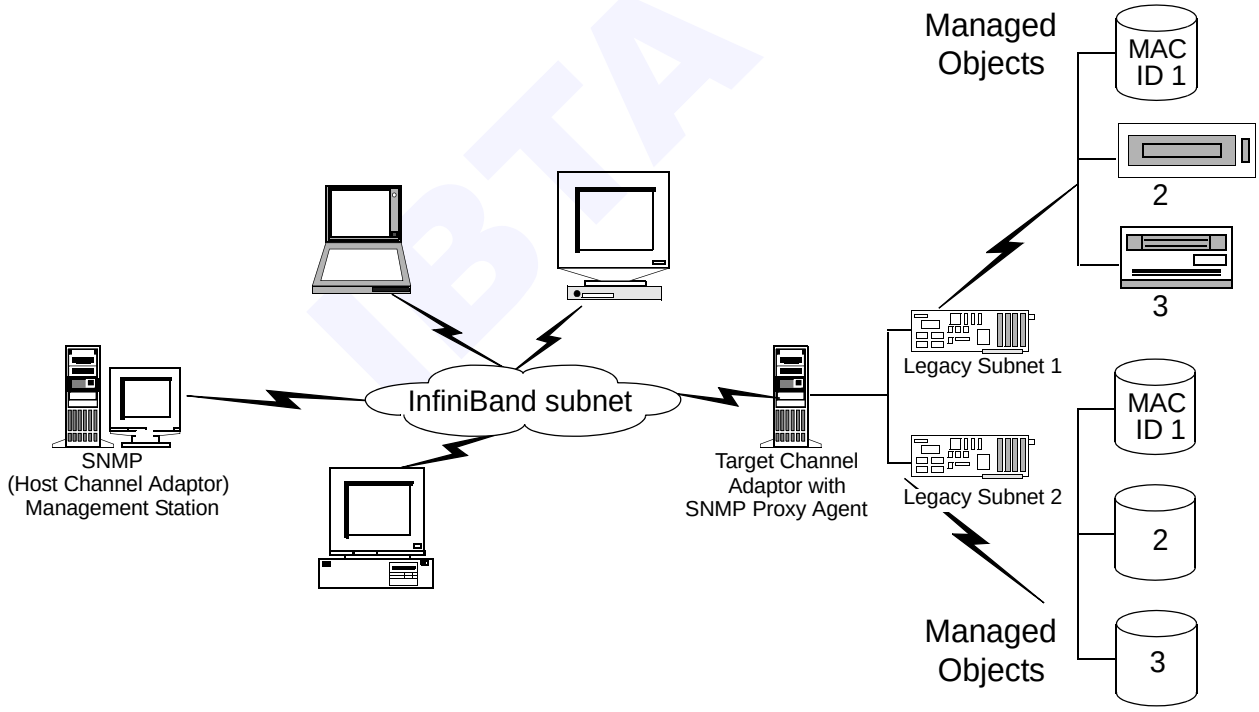


Figure 207 SNMP Proxy Agents

To target a remote managed object not directly connected to the InfiniBand fabric requires the use of an SNMP Proxy Agent. See Figure 207 SNMP Proxy Agents on page 1003. The basic function of a Proxy Agent is to receive SNMP packets passed up from the InfiniBand Endnode SNMP agent and forward them to that remote managed agent. These remote agents are, as such, not directly connected to the InfiniBand fabric and thus cannot be managed through it unless an intermediate device acts on its behalf to receive and send over the unsupported interconnect.

o16-11: The InfiniBand architecture **shall** be able to accommodate such legacy transports by redirecting SNMP packets destined for these managed nodes.

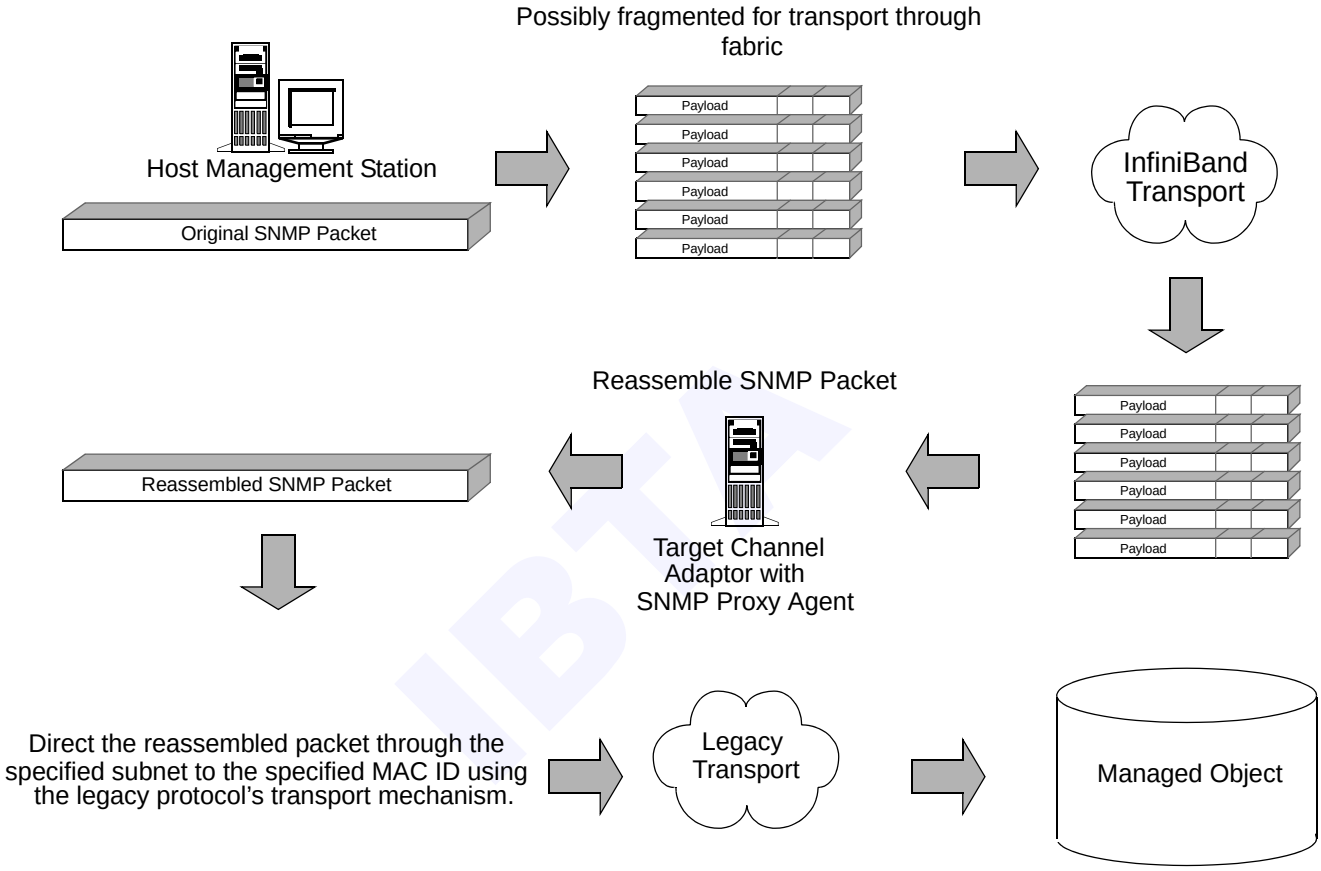


Figure 208 SNMP PDU Segmentation

SNMP targeting for beyond the InfiniBand Endnode (such as an InfiniBand device attached to a TCA that supports SNMP) is accomplished by an SNMP redirect. An SNMP packet destined for such a redirection will contain one of the SNMP redirect features and specify the destination address in the raddress field of the SNMP class datagram. This will allow the Proxy SNMP Agent to reassemble the SNMP packet from its fragments (if any) so it may re-encode the packet for the legacy transport over which it travels to reach its final destination.

16.4.4.2 TRAP EVENT SUBSCRIPTION

A node may request SNMP traps from a given node be sent to it. This is done by setting the ClassPortInfo Attribute with the LID and QP appropriately. The SNMP agent will transmit the Trap PDU as a sequence of

SNMP datagrams to the destination node. It is not using the method Trap() but the method Send() with the Trap PDU as the SNMP Data.

16.5 VENDOR-SPECIFIC

The Vendor-specific Agent is optional.

Vendor-specific operations can be defined using the vendor-specific management classes.

Vendors are free to define new methods and attributes and their use, provided that they conform to the common MAD format and methods defined herein, and do not conflict with the stated restrictions on method and attribute utilization. Vendor-specific classes will never be used to define management operations that are encompassed by the InfiniBand Architecture.

16.5.1 MAD FORMAT

o16-12: This optional compliance statement is obsolete and has been replaced by o16-12.1.1: and o16-12.1.2:.

The MADHeader:ClassVersion component for the Vendor class is subject to vendor versioning.

o16-12.1.1: For Vendor-Specific classes 0x09-0x0F, the datagrams in these Vendor-specific classes **shall** conform to the MAD format and use as specified in 13.4 Management Datagrams on page 717 and further customized in Figure 209 Vendor MAD Format (Classes 0x09-0x0F) on page 1005 and Table 294 Vendor MAD Fields (Classes 0x09-0x0F) on page 1005 below.

bytes	bits 31-24	bits 23-16	bits 15-8	bits 7-0
0	Common MAD Header			
...				
20				
24	Data			
...				
252				

Figure 209 Vendor MAD Format (Classes 0x09-0x0F)

Table 294 Vendor MAD Fields (Classes 0x09-0x0F)

Field Name	Length	Description
Common MAD Header	24 bytes	Common MAD Header as described in 13.4.2 Management Datagram Format on page 718

Table 294 Vendor MAD Fields (Classes 0x09-0x0F) (Continued)

Field Name	Length	Description
Data	232 bytes	The interpretation of the data is vendor-class specific.

o16-12.1.2: For Vendor-Specific classes 0x30-0x4F, the datagrams in these Vendor-specific classes **shall** conform to the MAD format and use as specified in 13.4 Management Datagrams on page 717 and further customized in Figure 210 Vendor MAD Format (Classes 0x30-0x4F) on page 1006 and Table 295 Vendor MAD Fields (Classes 0x30-0x4F) on page 1006 below.

bytes	bits 31-24	bits 23-16	bits 15-8	bits 7-0
0	Common MAD Header			
...				
20				
24	RMPP Header			
28				
32				
36	Reserved	OUI		
40	Data			
...				
252				

Figure 210 Vendor MAD Format (Classes 0x30-0x4F)

Table 295 Vendor MAD Fields (Classes 0x30-0x4F)

Field Name	Length	Description
Common MAD Header	24 bytes	Common MAD Header as described in 13.4.2 Management Datagram Format on page 718
RMPP Header	12 byte	RMPP Header as described in 13.6.2.1 RMPP Header on page 772
Reserved	1 byte	Reserved
OUI	3 bytes	IEEE assigned Vendor OUI
Data	228 bytes	The interpretation of the data is vendor-class specific.

16.5.2 STATUS FIELD

The Status field is described in 13.4.7 Status Field on page 731. Class-specific bits are defined by the Vendor.

Table 296 Vendor Status Field

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.7 Status Field on page 731
8-15	-	Class-specific bits defined by Vendor

16.5.3 METHODS

Vendor classes support the common methods.

Table 297 Vendor Class Methods

Method Type	Value	Description
VendorGet()	0x01	Request an attribute to return (read) from a target.
VendorSet()	0x02	Request a target to set (write) an attribute. The object will issue a VendorGetResp() as a response.
VendorGetResp()	0x81	Target responds to an attribute Get()/Set() request.
VendorSend()	0x03	Send a datagram. Does not require a response.
VendorTrap()	0x05	Unsolicited datagram sent to the vendor entity. Contains the Notice Attribute as defined in 13.4.8.2 Notice on page 737 to identify the trap.
VendorTrapRepress()	0x07	Block repetition of notification.

Vendor-specific methods can be added as desired by vendor, providing there is no collision with reserved methods in 13.4.5 Management Class Methods on page 721.

16.5.4 ATTRIBUTES

o16-13: This optional compliance statement is obsolete and has been deleted.

The Vendor classes may optionally support the attributes Notice and InformInfo. All other attributes are vendor-defined.

Table 298 Vendor Class Attributes

Attribute Name	Attribute ID	Attribute-Modifier	Description
ClassPortInfo	0x0001	0x00000000	See 16.5.4.1 ClassPortInfo on page 1008

Table 298 Vendor Class Attributes (Continued)

Attribute Name	Attribute ID	Attribute-Modifier	Description
Vendor defined	0x0010 - 0xFFFF	0x00000000-0xFFFFFFFF	

Table 299 Vendor Attribute / Method Map

Attribute Name	VendorGet()	VendorSet()	VendorSend()	VendorTrap()
ClassPortInfo	X	X (if traps defined)		
Vendor defined	Vendor defined			

16.5.4.1 CLASSPORTINFO

The ClassPortInfo attribute is described in 13.4.8.1 ClassPortInfo on page 734. Class-specific bits are defined by Vendor.

Table 300 Vendor ClassPortInfo:CapabilityMask

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.8.1 ClassPortInfo on page 734
8-15	-	Class-specific bits defined by Vendor

16.6 APPLICATION-SPECIFIC

The Application-specific Agents are optional.

Application-specific operations can be defined using the application-specific management classes.

Applications are free to define new methods and attributes and their use, provided that they conform to the common MAD format and methods defined herein, and do not conflict with the stated restrictions on method and attribute utilization. Application-specific classes may be used to define application management operations that are encompassed by the InfiniBand Architecture. Vendors should not use application-specific classes to define vendor-specific behavior; instead vendor-specific management classes should be used.

16.6.1 MAD FORMAT

o16-14: The datagrams in these Application-specific classes **shall** conform to the MAD format and use as specified in 13.4 Management Datagrams on page 717 and further customized in Figure 211 Application MAD

Format on page 1009 and Table 301 Application MAD Fields on page 1009 below.

bytes	bits 31-24	bits 23-16	bits 15-8	bits 7-0
0	Common MAD Header			
...				
20				
24	Data			
...				
252				

Figure 211 Application MAD Format

Table 301 Application MAD Fields

Field Name	Length	Description
Common MAD Header	24 bytes	Common MAD Header as described in 13.4.2 Management Datagram Format on page 718
Data	232 bytes	Interpretation of the data is application-class specific

16.6.1.1 STATUS FIELD

The Status field is described in 13.4.7 Status Field on page 731. Class-specific bits are defined by the Application.

Table 302 Application Status Field

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.7 Status Field on page 731
8-15	-	Class-specific bits defined by Application

16.6.2 METHODS

Application classes support the common methods.

Table 303 Application Class Methods

Method Type	Value	Description
AppGet()	0x01	Request an attribute to return (read) from a target.
AppSet()	0x02	Request a target to set (write) an attribute. The object will issue a AppGetResp() as a response.
AppGetResp()	0x81	Target responds to an attribute Get()/Set() request.
AppSend()	0x03	Send a datagram. Does not require a response.

Table 303 Application Class Methods (Continued)

Method Type	Value	Description
AppTrap()	0x05	Unsolicited datagram sent to the application entity. Contains the Notice Attribute as defined in 13.4.8.2 Notice on page 737 to identify the trap.
AppTrapRepress()	0x07	Block repetition of notification.

Application-specific methods can be added as desired by application, providing there is no collision with reserved methods in 13.4.5 Management Class Methods on page 721.

16.6.3 ATTRIBUTES

o16-15: This optional compliance statement is obsolete and has been deleted.

The Application classes may optionally support the attributes Notice and InformInfo. All other attributes are application-defined.

Table 304 Application Class Attributes

Attribute Name	Attribute ID	Attribute-Modifier	Description
ClassPortInfo	0x0001	0x00000000	See 16.6.3.1 ClassPortInfo on page 1010
Application defined	0x0011 - 0xFFFF	0x00000000-0xFFFFFFFF	

Table 305 Application Attribute / Method Map

Attribute Name	AppGet()	AppSet()	AppSend()	AppTrap()
ClassPortInfo	X	X		
Application defined	Application defined			

16.6.3.1 CLASSPORTINFO

The ClassPortInfo attribute is described in 13.4.8.1 ClassPortInfo on page 734. Class-specific bits are defined by Application.

Table 306 Application ClassPortInfo:CapabilityMask

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.8.1 ClassPortInfo on page 734
8-15	-	Class-specific bits defined by Application

16.7 COMMUNICATION MANAGEMENT

C16-14: This compliance statement is obsolete and has been deleted.

Communication Management is described in Chapter 12: Communication Management on page 650. The Communication Management functions required for nodes are described in that chapter. Proper use of the messages defined in this section is subject to the protocols and state machines defined in that chapter.

16.7.1 MAD FORMAT

C16-15: The datagrams in the Communication Management class **shall** conform to the MAD format and use as specified in 13.4 Management Datagrams on page 717 and further customized in Figure 212 Communication Management MAD Format on page 1011 and Table 307 Communication Management MAD Fields on page 1011 below.

bytes	bits 31-24	bits 23-16	bits 15-8	bits 7-0
0	Common MAD Header			
...				
20				
24	Data			
...				
252				

Figure 212 Communication Management MAD Format

Table 307 Communication Management MAD Fields

Field Name	Length	Description
Common MAD Header	24 bytes	Common MAD Header as described in 13.4.2 Management Datagram Format on page 718
Data	232 bytes	Attribute data is mapped bit for bit from the format described in the following sections to the start of this data field. If the attribute is smaller than the data field, the content of the remainder of the data field is unspecified.

C16-15.1.1: The MADHeader:ClassVersion component for the CM class **shall** be 2 for this version of the specification.

16.7.1.1 STATUS FIELD

The Status field is described in 13.4.7 Status Field on page 731. No class-specific bits are defined.

Table 308 Communication Management Status Field

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.7 Status Field on page 731
8-15	-	Reserved (class-specific bits)

16.7.2 METHODS

The Communication Management class supports the methods identified in Table 309 Communication Management Methods on page 1012 below.

C16-15.1.2: The Communication Management Class **shall** support the methods listed in Table 309: Communication Management Methods. All method type values not listed in the Table are reserved.

Table 309 Communication Management Methods

Method Type	Value	Description
ComMgtGet()	0x01	Request a get (read) of an attribute
ComMgtSet()	0x02	Request a set (write) of an attribute.
ComMgtGetResp()	0x81	Response from a Get() or Set() request.
ComMgtSend()	0x03	Send a connection management message.

16.7.3 ATTRIBUTES

The Attributes/AttributeModifiers specified in this section describe the mappings of message parameters defined in Chapter 12: Communication Management on page 650 into the standard MAD header/payload format. The set of attributes supported by the Communication Management class is listed Table 310 Communication Management Attributes on page 1013

C16-15.1.3: The Communication Manager **shall** support the attributes and methods listed in Table 310 Communication Management Attributes on page 1013 and Table 311 Communication Management Attribute /

Method Map on page 1013. All attribute IDs not listed in Table 310: Communication Management Attributes are reserved.

Table 310 Communication Management Attributes

Attribute Name	Attribute ID	Attribute-Modifier	Description
ClassPortInfo	0x0001	0x00000000	Refer to 13.4.8.1 ClassPortInfo on page 734
ConnectRequest	0x0010	0x00000000	Refer to 12.6.5 REQ - Request for Communication on page 659
MsgRcptAck	0x0011	0x00000000	Refer to 12.6.6 MRA - Message Receipt Acknowledgment on page 661
ConnectReject	0x0012	0x00000000	Refer to 12.6.7 REJ - Reject on page 662
ConnectReply	0x0013	0x00000000	Refer to 12.6.8 REP - Reply to Request for Communication on page 668
ReadyToUse	0x0014	0x00000000	Refer to 12.6.9 RTU - Ready To Use on page 669
DisconnectRequest	0x0015	0x00000000	Refer to 12.6.10 DREQ - Request for communication Release (Disconnection REQuest) on page 669
DisconnectReply	0x0016	0x00000000	Refer to 12.6.11 DREP - Reply to Request for communication Release on page 670
ServiceIDResReq	0x0017	0x00000000	Refer to 12.11.1 SIDR_REQ - Service ID Resolution Request on page 706
ServiceIDResReqResp	0x0018	0x00000000	Refer to 12.11.2 SIDR_REP - Service ID Resolution Response on page 707
LoadAlternatePath	0x0019	0x00000000	Refer to 12.8.1 LAP - Load Alternate Path on page 681
AlternatePathResponse	0x001A	0x00000000	Refer to 12.8.2 APR - Alternate Path Response on page 682

Table 311 Communication Management Attribute / Method Map on page 1013 indicates the methods with which each of the attributes is valid.

Table 311 Communication Management Attribute / Method Map

Attribute	ComMgtGet()	ComMgtSet()	ComMgtSend()
ClassPortInfo	X		
ConnectRequest			X
ConnectReply			X
ReadyToUse			X
MsgRcptAck			X
ConnectReject			X

Table 311 Communication Management Attribute / Method Map (Continued)

Attribute	ComMgtGet()	ComMgtSet()	ComMgtSend()
DisconnectRequest			X
DisconnectReply			X
LoadAlternatePath			X
AlternatePathResponse			X
ServiceIDResReq			X
ServiceIDResReqResp			X

The normative definitions of the attribute components and the operational requirements and constraints applicable thereto are defined in Chapter 12: Communication Management on page 650.

16.7.3.1 CLASSPORTINFO

The ClassPortInfo attribute is described in 13.4.8.1 ClassPortInfo on page 734. In addition, bits 8 through 12 of the CapabilityMask component are defined:

Table 312 Communication Management ClassPortInfo:CapabilityMask

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.8.1 ClassPortInfo on page 734
8	Reserved	Reserved
9	IsReliableConnectionCapable	The CM associated with this port supports the establishment and release of connections between Reliable Connected Queue Pairs. It will accept and respond to all mandatory messages as defined in 12.6.1 Required Messages on page 655.
10	IsReliableDatagramCapable	The CM associated with this port supports the establishment and release of reliable datagram channels between EE Contexts. It will accept and respond to all mandatory messages as defined in 12.6.1 Required Messages on page 655.
11	Undefined	Undefined; any use of this field is vendor-dependent.

Table 312 Communication Management ClassPortInfo:CapabilityMask (Continued)

Bits	Name	Meaning
12	IsUnreliableConnectionCapable	The CM associated with this port supports the establishment and release of connections between Unreliable Connected Queue Pairs. It will accept and respond to all mandatory messages as defined in 12.6.1 Required Messages on page 655.
13	IsSIDRCapable	The CM associated with this port will respond to incoming Service ID Resolution requests as defined in 12.11 Service ID Resolution Protocol on page 705.
14-15	Reserved	Reserved

CHAPTER 17: CHANNEL ADAPTERS

17.1 OVERVIEW

This section specifies the minimum requirements for an IBA channel adapter. Channel adapters (CA) are the source and terminus of IBA packets that traverse the IBA switching fabric. Channel adapters are either Host Channel Adapters (HCAs) or Target Channel Adapters (TCAs). In a typical system, the HCAs are used by the host processors to connect to the IBA fabric whereas the TCAs are used by an I/O adapter to connect to the IBA fabric.

The key difference between a Host Channel Adapter and a Target Channel Adapter is in the way the client (whether the client is hardware or software) interfaces to the transport layer. Specifically, the HCA supports the IBA Verbs layer whereas the TCA uses an implementation dependent interface to the transport layer.

C17-1: An HCA shall support the IBA verbs layer interface.

Previous sections of the specification have described the various layers comprising an IBA Channel Adapter (physical, link, network, and transport layers). All channel adapters share a common architecture for the physical, link, network and transport layers. See Figure 213 below. From the point of view of the physical communications link, an HCA and TCA are identical.

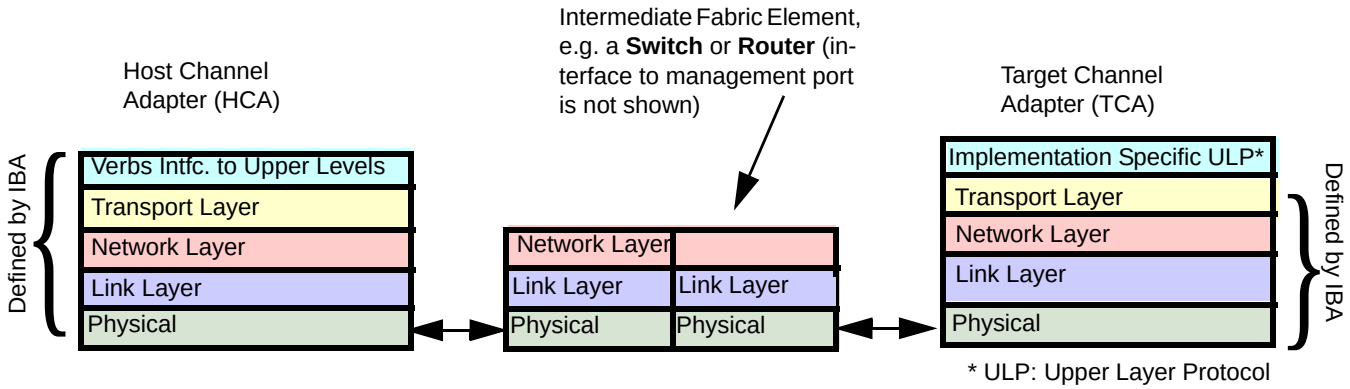


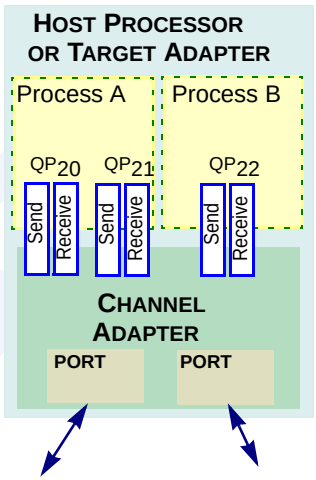
Figure 213 IBA Architecture Layers

This chapter lists the common functionality in all CAs as well as the differences between HCAs and TCAs. There are also differences in required minimum functionality. These issues are addressed in the following sections.

17.2 COMMON FUNCTIONAL REQUIREMENTS

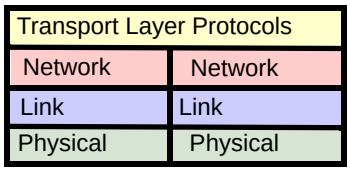
17.2.1 MULTIPLE PORTS PER CHANNEL ADAPTER

A Channel Adapter[25] may have one or more ports. A CA's port provides the physical, link and network protocol layers of the IBA CA. A channel adapter with multiple ports shares the transport layer functionality amongst the ports. For example, a QP (a transport layer construct) can be configured to work with any of the ports on the CA. The following figures show the physical representation as well as the architectural layering.



A two ported Channel Adapter. The CA has one block of QPs, memory translation tables etc. that interact with the IBA fabric through the two ports.

Figure 214 Multiport CA



A two ported CA showing a common transport layer and independent Network, Link and Physical Layers for each port.

Figure 215 Multiport CA Architectural Layers

It is desirable for a channel adapter to have multiple ports for several reasons:

25. Unless specifically mentioned, the term Channel Adapter refers to both an HCA and a TCA.

- **Increased bandwidth from a single CA**. e.g. an HCA with a high performance host memory interface can support the bandwidth of several IBA links. By adding relatively low cost IBA ports the HCA can multiply its throughput with relatively little additional cost. See Figure 217 on page 1019.
- **Redundant paths for fault tolerant communications**. In a system with multiple paths between source and destination, a CA's multiple ports may be used to tolerate faults in the fabric's switches and links. See Figure 218 on page 1019 and Figure 219 on page 1020.
- **Support direct links to TCAs.** in a low cost, switchless topology the ports of an HCA might be directly wired to TCAs. See Figure 220 on page 1020.

17.2.1.1 TOPOLOGIES SUPPORTED WITH MULTI-PORTED CHANNEL ADAPTERS

The following diagrams show several basic ways a multi ported CA could be attached to the rest of the system. These diagrams are by no means the only topologies supported -- they are examples only. Note that multiple ports on a CA may either connect to multiple subnets or to the same subnet.

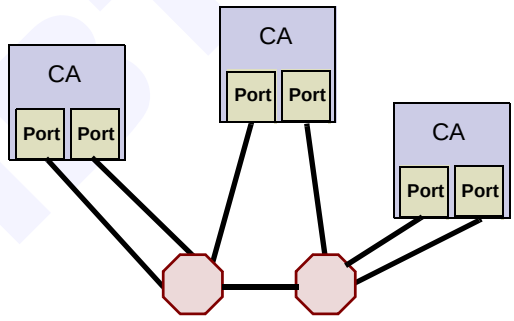
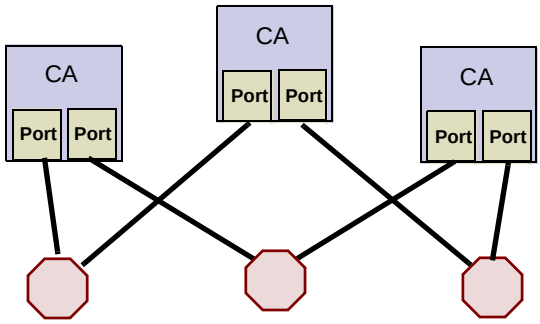


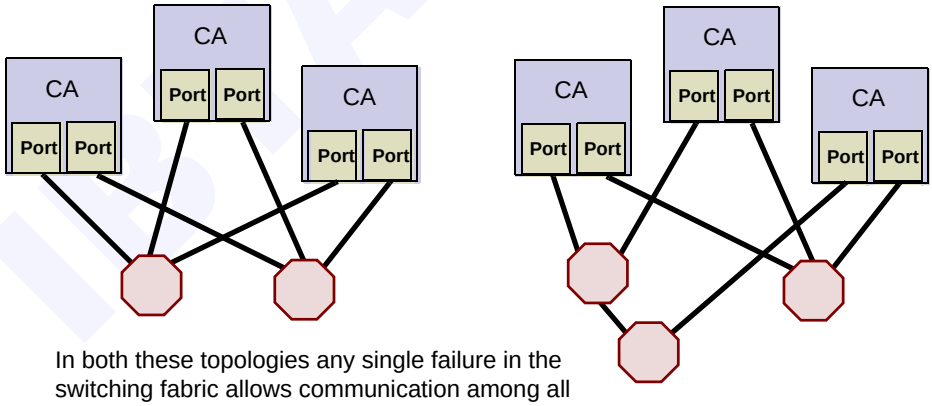
Figure 216 Multiported CAs Connected to Single Subnet

C17-2: A multiported CA shall be capable of connecting to one or more subnets.



In this topology not all ports can reach all destination CAs. In this diagram there are three separate subnets.

Figure 217 Multiported CAs Connected to Multiple Subnets



In both these topologies any single failure in the switching fabric allows communication among all CAs to continue.
Two cases are shown: symmetric fabrics and non identical fabrics

Figure 218 Fault Tolerant Connections to Independent Fabrics

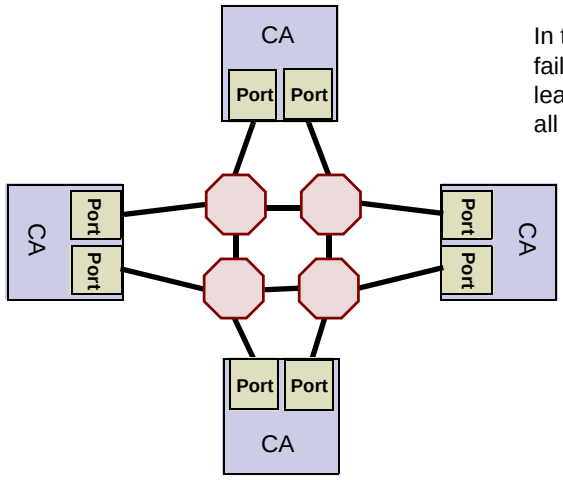


Figure 219 Fault Tolerant Connection to a Single Fabrics

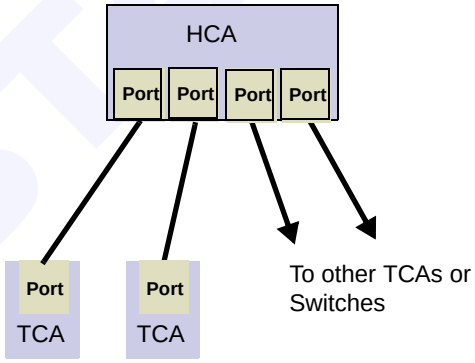


Figure 220 Multiported HCA with Direct Connections to TCAs

17.2.1.2 ASSOCIATION OF QPS WITH PORTS

C17-3: While a CA may have many QPs and many ports, each QP shall generate request packets, service returning response packets, and respond to arriving request packets through exactly one port, at any point in time.

To ensure packet ordering within a QP for connected or reliable transport services, packets are required to take the same path between a source and destination. This requires that all requests and responses use a consistent port, base SLID, DLID, VL and SL. A connected or reliable transport QP remains bound with one port until path migration for error recovery or load balancing purposes occurs or the connection goes away.

Aside from valid packets requesting path migration as described in Section 17.2.8 Automatic Path Migration on page 1031, incoming request and response packets arriving at a port other than the port currently bound to the appropriate QP may be discarded.

For an HCA using the unreliable datagram transport service, the verbs layer specifies the remote address with each outgoing work request. Since the QP is only bound to one port, the client of the verbs layer must be certain the destination is reachable from that port. In certain topologies not all destinations are reachable from all ports (see Figure 217 on page 1019).

Incoming Unreliable Datagram packets may only target a QP if that QP is bound to the port on which the datagram arrived.

The Reliable Datagram service uses an end-to-end context to ensure correct delivery for every channel adapter with which it communicates. The EE context, like a Reliable Connection QP, is bound to one port (at least until path migration is used to rebind the EE context with a new port). But since a RD QP can communicate with multiple EE contexts, the RD QP can in effect be transmitting and receiving packets from multiple ports.

17.2.1.3 PORT ATTRIBUTES AND FUNCTIONS

Certain attributes and functions are associated with each port. Typically these belong to the physical, link, and network layers that are unique to the port. The table below itemizes these as well as describes some transport layer functionality unique to each port. Each attribute or function is intended to be applied individually to each port.

Table 313 Port Attributes & Functions

Attribute/Function	HCA	TCA
Physical Interface	The HCA and TCA shall support one or more of the IBA defined physical interfaces. (See the IBA Specification, Volume 2)	
Static Rate Control (limiting the BW to a particular destination CA)	required on ports supporting bandwidths above 2.5 Gbps	
Support for multipathing (see Section 7.11 Subnet Multipathing on page 219)	required	required

Table 313 Port Attributes & Functions (Continued)

Attribute/Function	HCA	TCA
P_Key Checking on inbound Request and Inbound Response Packets (see Section 10.9 Partitioning on page 523)	Shall validate incoming packet's P_Key with the P_Key bound to the destination QP[a]. A CA shall maintain a P_Key table (see Section 10.9.2 The Partition Key Table (P_Key Table) on page 525) per port. Each table shall have at least one P_Key entry. HCA requires no OS involvement to set the P_Key (i.e. P_Key is set directly by a Subnet Manager control packet.)	
Validation of incoming packet's DLID and, if the GRH is present, DGID	required	required
Support for QP0 and QP1	required on each port	required on each port
Port Numbering	Ports are numbered starting from one and if there are multiple ports, they are numbered sequentially. MADs use port number zero as a wild-card port number that matches whatever port the packet arrived at. See 14.2.5.6 PortInfo on page 821.	
GID Support	Each port has at least one GID. The maximum number of GIDs per port is implementation specific. See the discussion on GIDs in Section 4.1 Terminology And Concepts on page 142.	

a. excluding Raw Datagram QPs (because raw datagrams don't have a P_Key). Also PKey checking for QP0 and QP1 are slightly different. See Section 10.9.8 Partition Enforcement on Management Queue Pairs on page 529 for more information.

C17-4: Static rate controls, as listed in Section 17.2.6 Static Rate Control on page 1029, are required on each port that supports a data rate above 2.5 gbps.

C17-5: Each port shall validate the incoming P_Key in an IB Transport packet with the P_Key bound to the destination QP (other than QP0 and QP1).

C17-6: The CA shall maintain a P_Key table per port supporting at least one and at most 65,535 P_Key entries.

C17-7: An HCA shall require no OS involvement to set the P_Key table; the P_Key table shall be set directly by Subnet Manager MADs.

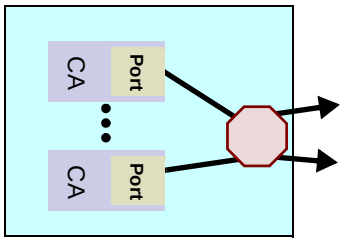
C17-7.a1: A CA may support up to 254 ports. For a CA supporting N ports, the ports shall be numbered from 1 to N.

C17-8: Each port shall support at least one GID.

17.2.1.4 SWITCHING PACKETS THROUGH MULTIPLE PORTS

If a Channel Adapter has multiple ports, the CA does not route packets from one port to the other. Such a packet forwarding function is defined as a switch.

An implementation may choose to package a switch and multiple IBA ports together, as shown in the figure below.



The overall box shows the boundary of the combined switch and Channel Adapter. This boundary could be a single IC or board. The boundary also represents the fault zone for that device.

Figure 221 Multiple Single Ported CAs with an Embedded Switch

17.2.2 CHANNEL ADAPTER ATTRIBUTES

The previous section described attributes of a channel adapter's ports. This section describes attributes of the whole channel adapter.

This specification only sets the minimum functionality of an HCA or TCA. For example, only two QPs are required (both for management). A practical HCA or TCA would undoubtedly support more QPs, but this section only specifies architectural minimum requirements. The following summa-

rizes various required and optional Channel Adapters attributes (see also section 11.2.1.2 Query HCA on page 551).

Table 314 Channel Adapter Attributes

Attribute	HCA	TCA
Support for multiple ports.	Optional	Optional
Source/Sink packets with a LRH (for communication within the subnet)	Required for all QPs.	
Source/Sink packets with a GRH (for communication across subnets)	Required for all QPs other than QP0.	
Transport Services Supported	HCAs shall be capable of supporting the Unreliable Datagram, Reliable Connection, and Unreliable Connection transport service on any QP supported by the HCA.	Aside from supporting Unreliable Datagram for the two required management QPs, support for any other transport service (or QP) is optional.

Table 314 Channel Adapter Attributes (Continued)

Attribute	HCA	TCA
Atomic Operations Supported	Optional to generate requests or responses	
Other Operations Supported	If a transport service is supported, then the CA must support all the operations defined for that transport service (excluding atomic operations).	TCAs are not general purpose and may customize the operations supported to suit their function (e.g. a TCA with Reliable Connection Service may generate RDMA requests but not respond to incoming RDMA requests)
Solicited Events (see Section 9.2.3 Solicited Event (SE) - 1 bit on page 238 and Section 11.4.2.2 Request Completion Notification on page 627	Required to both generate solicited events and to receive them.	Optional
MTU	CAs shall support one of the following sets of MTUs (for all Transport Service Classes): • 256 Bytes • 256, 512 Bytes • 256, 512, 1024 Bytes • 256, 512, 1024, 2048 Bytes • 256, 512, 1024, 2048, 4096 Bytes	
	For UD and Raw the WQE determines the packet size. The maximum packet size is limited by MTUCap and NeighborMTU. For RD the EE context specifies the MTU. For RC and UC the QP specifies the MTU.	Selection of MTU for TCAs is implementation specific.
End-to-End Flow Control (reliable connection transport service only)	HCA receive queues that are not associated with an SRQ shall generate E-to-E flow control credits • i.e. HCAs throttle inbound requests to prevent inbound Sends arriving at an empty receive queue. HCA receive queues that are associated with an SRQ shall not generate E-to-E flow control credits • i.e. HCAs shall not throttle inbound requests HCA send queue shall receive and respond to inbound credits • i.e. remote node may throttle the HCA's outbound requests.	TCA receive queues may generate E-to-E flow control credits. • i.e. TCA need not throttle inbound requests. TCA send queue shall receive and respond to inbound credits. • i.e. remote node may throttle the TCA's outbound requests.

Table 314 Channel Adapter Attributes (Continued)

Attribute	HCA	TCA
Multicast	Generating IBA Raw Multicast packets is optional. Receiving IBA Raw Multicast packets is optional. Generating IBA Unreliable Datagram Multicast packets is optional[a]. Receiving IBA Unreliable Datagram Multicast packets is optional.	
Automatic Path Migration	It is optional to either generate or respond to an automatic path migration request.	
Memory Protection	HCA's provide memory protection as described in Section 10.6 Memory Management on page 468.	Optional
Loopback Support	Self addressed packets[b] shall be allowed and shall not go out onto the wire. That is, self addressed packets must work even if no external switch is present	Optional

a. It is expected that any implementation of the Unreliable Datagram transport service will trivially support the generation of multicast packets.
b. A self-addressed packet is a packet whose DLID and SLID (while not necessarily identical) address the same port of the same CA. A self-addressed packet may or may not have the same source and destination QP. IB does not define a specific "loopback" address.

C17-9: All channel adapters shall be able to source and sink (to all QPs) locally routed packets (i.e. no GRH).

C17-10: All channel adapters shall be able to source and sink (to all QPs other than QP0) globally routed packets (i.e. packets with a GRH).

C17-11: HCAs shall be capable of supporting the Unreliable Datagram, Reliable Connection, and Unreliable Connection transport service on any QP supported by the HCA.

C17-12: If a transport service is supported by an HCA, then that HCA must support all the operations defined for that transport service (excluding atomic operations).

C17-13: An HCA shall be able to generate and receive solicited event.

C17-14: CAs shall support one of the following sets of MTUs (for all Transport Service Classes):
256 Bytes
256, 512 Bytes
256, 512, 1024 Bytes
256, 512, 1024, 2048 Bytes
256, 512, 1024, 2048, 4096 Bytes

C17-15: This compliance statement has been obsoleted and replaced by C17-15.2.1:.

C17-15.2.1: HCA receive queues that are not associated with an SRQ, shall generate E-to-E flow control credits. HCA receive queues that are associated with an SRQ, shall not generate E-to-E flow control credits.

C17-16: HCA send queue shall receive and respond to inbound E-to-E flow control credits.

o17-1: This compliance statement has been obsoleted and replaced by o17-1.2.1:.

o17-1.2.1: TCA receive queues that are not associated with an SRQ may generate E-to-E flow control credits. TCA receive queues that are associated with an SRQ, shall not generate E-to-E flow control credits.

C17-17: TCA send queue shall receive and respond to inbound E-to-E flow control credits.

o17-2: A CA may be capable of generating multicast packets.

o17-3: A CA may be capable of receiving multicast packets.

o17-4: A CA may be capable of generating and responding to the Automatic Path Migration protocol.

C17-18: HCAs shall allow packets with a destination address the same as that of the port on which the packet is issued. Such a loopback packet shall not go onto the wire.

17.2.3 DEADLOCK PREVENTION

Each CA shall not cause deadlock in the fabric. This condition is met by

- The CA will not continuously and permanently assert backpressure (i.e. fail to grant link credits).
- The CA shall not assert backpressure on a port's inbound link as the result of receiving backpressure on that port's outbound link.

C17-19: For deadlock prevention, the CA shall not continuously and permanently assert backpressure (i.e. fail to grant link credits).

C17-20: For deadlock prevention, the CA shall not assert backpressure on a port's inbound link as the result of receiving backpressure on that port's outbound link.

17.2.4 CHECKING INCOMING PACKETS

All CA's are required to validate each incoming packet before committing the packet to the CA's state.

C17-21: The CA shall check for link, network and transport layer errors in all incoming packets.

17.2.5 NON-VOLATILE STATE

C17-22: All channel adapters shall maintain a EUI-64 port GUID and a EUI-64 CA GUID (See Chapter 4: Addressing on page 141) in nonvolatile memory such that the GUID is the same each time the CA is powered on.

C17-23: This compliance statement is obsolete and has been removed

Other uses of the nonvolatile memory are optional.

The type of non volatile memory in a CA is not specified and might be a local disk drive or on-chip memory.

IBA does not require a CA to remember connection state information across power cycles.

17.2.6 STATIC RATE CONTROL

A CA shall support static rate control (see section 9.11 Static Rate Control on page 427) if its raw bandwidth is greater than 2.5 Gbps. The Inter-packet Delay (IPD) values supported (see Table 63 on page 428) must allow slowing the packet rate to all of the standard link rates. The table below indicates the values of IPD that shall be supported

Table 315 Static Rate Control IPD Values

PortInfo:LinkWidthSupported	PortInfo:LinkSpeedSupported	Required IPDs	Optional IPDs[a]
1	1	0	-
1	3	0, 1	-
1	5 or 7	0, 1, 3	-
3	1	0, 3	1
3	3	0, 1, 3, 7	-

Table 315 Static Rate Control IPD Values (Continued)

PortInfo:LinkWidthSupported	PortInfo:LinkSpeedSupported	Required IPDs	Optional IPDs[a]
3	5 or 7	0, 1, 3, 7, 15	-
7	1	0, 1, 3, 7	-
7	3	0, 1, 3, 7, 15	-
7	5 or 7	0, 1, 2, 3, 7, 15, 31	-
11	1	0, 2, 3, 11	1, 5, 7
11	3	0, 1, 2, 3, 5, 7, 11, 15, 23	-
11	5 or 7	0, 1, 2, 3, 5, 7, 11, 15, 23, 31, 47	-
15	1	0, 1, 2, 3, 5, 7, 11	-
15	3	0, 1, 2, 3, 5, 7, 11, 15, 23	-
15	5 or 7	0, 1, 2, 3, 5, 7, 11, 15, 23, 31, 47	-

a. Support for these optional values is indicated by setting the PortInfo:CapabilityMask:IsOptionalIPDSupported bit.

Note, the optional values are to grandfather implementations done prior to the Architectural Release 1.2. New implementations should support these optional IPD values.

17.2.7 MANAGEMENT MESSAGES

Each port of every channel adapter shall support two QPs for management commands:

- QP0, used by the Subnet Management Agent for sending and receiving Subnet Management Packets (SMPs).
  - This QP uses the Unreliable Datagram transport service.
  - SMP packets arriving before the current packet's command completes may be dropped (i.e. the minimum queue depth of QP0 is one).
- QP1, used for the General Services Interface (GSI).
  - This QP uses the Unreliable Datagram transport service.
  - All traffic to and from this QP uses any VL other than VL15.
  - GSI packets arriving before the current packet's command completes may be dropped (i.e. the minimum queue depth of QP1 is one).

C17-24: Each port of every CA shall support QP0 for use by the SMA and QP1 for use by the GSA.

All QPs for a given CA, except QP0 and QP1 have unique numbers. QP0 and QP1 are special in that each port has its own QP0 and QP1.

The rest of the (non RD) QPs on a CA may be bound with any one port. The binding of a QP (other than QP0 or QP1) with a port is maintained until such time that automatic path migration (see 17.2.8 Automatic Path Migration on page 1031) or path migration requested by MADs associates the QP with a different port.

The management QPs are special because they are used by the Subnet Manager and other management applications. See Section 13.5.1 MAD Interfaces on page 749.

Since each port may be on a different subnet it must communicate with a different Subnet Manager and related management application, The Subnet Manager and other nodes using the GSI use the well known QP numbers (0 and 1) to establish communication.

17.2.7.1 SUBNET MANAGEMENT

All CAs shall respond to incoming Subnet Management Packets from the Subnet Manager. CAs shall also generate the required traps defined as part of the SMA.

The IBA does not require nor preclude a CA from being a Subnet Manager. If a node does host a Subnet Manager, it must meet the requirements as specified in section 14.4 Subnet Manager on page 859.

17.2.7.2 GENERAL SERVICES

All CAs shall respond to mandatory GSI MADs defined in Chapter 16: General Services on page 930. Any HCA or TCA may initiate MADs to another CA.

17.2.8 AUTOMATIC PATH MIGRATION

The reliable or connected transport services (Reliable Connection, Reliable Datagram, and Unreliable Connection) use the same path for a given connection (or in the case of RD for a given pair of end-to-end contexts). This ensures data is delivered in the proper order. Path migration refers to the requestor and responder agreeing to use a new path. The source and destination QPs remain the same but the ports and path through the fabric may change. Path migration may be used to recover from a bad path (sometimes this is referred to as Failover) or for other reasons such as load balancing.

Automatic path migration may be supported by HCAs and TCAs. If supported, Automatic path migration works for QPs using the RC, RD, and UC transport services.

Automatic path migration provides a fast mechanism for path migration. When a connection is established the two CA's use Communication Management MADs to establish the primary and alternate path (See sections 10.4 Automatic Path Migration on page 461 and 12.8 Alternate Path Management on page 680. Automatic path migration is a mechanism whereby either CA can signal the other to switch from the primary path to the alternate path.

At connection establishment time the CA is set with the following information to determine a path:

- DLID of the responding CA
- Destination GID of the responding CA
- SL
- source port (i.e. the base SLID and path bits for outbound request and response packets)

At connection establishment the CAs may be given two sets of path information, one for the primary path and another for the alternate path. The alternate path may use the same or different source and destination ports as that used by the primary path.

17.2.8.1 AUTOMATIC PATH MIGRATION PROTOCOL

The automatic path migration protocol uses a single bit in the BTH called MigReq (Migrate Request) and a 3-state state machine associated with each connected QP supporting automatic path migration. If automatic path migration is not supported by either QP of the connection, the state

machines of the two QP's remain in the MIGRATED state.See figure below.

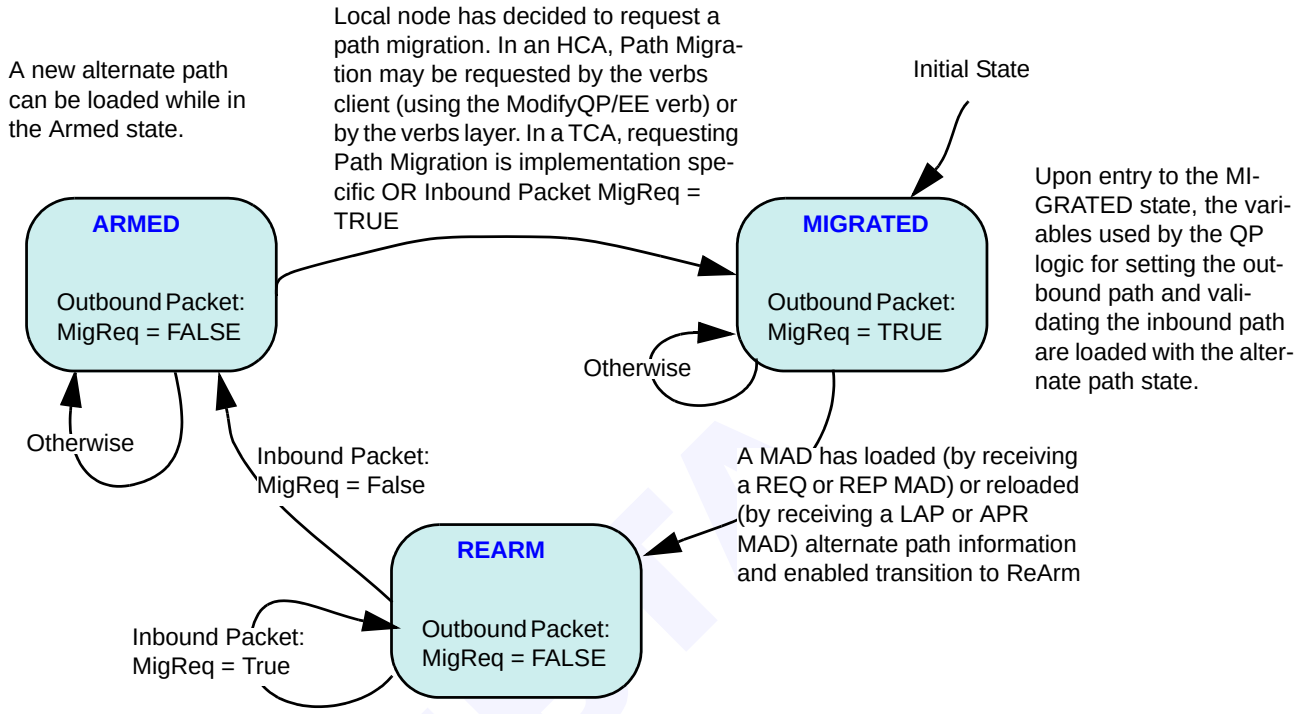


Figure 222 Automatic Path Migration State Machine (per QP)

17.2.8.1.1 INITIALIZATION

At connection setup time, the primary and alternate path states are assigned to each CA.

The Automatic Path Migration State Machine is initialized to the MIGRATED state whether or not Automatic Path Migration is supported by either QP of the connection.

17.2.8.1.2 MIGRATION REQUEST

Either CA may request an automatic path migration. Reasons for requesting automatic path migration are outside the scope of the IBA specification but may include load balancing or using an alternate path to recover from excessive errors.

The CA requesting automatic path migration transitions its state machine to the MIGRATED state. Once in the MIGRATED state the CA generates all new packets (both request and response packets) using the path that was previously initialized as the alternate path. The CA may refuse to accept incoming request or response packets arriving from the original path.

Once in the MIGRATED state, all outbound packets (both requests and responses) on that QP have the MigReq (Migrate Request) bit in the BTH set to TRUE.

17.2.8.1.3 MIGRATION RESPONSE

A CA whose QP is in the ARMED state that receives a packet (either request or response packet) with the MigReq bit set validates the incoming packets path bits with the alternate path information (i.e. checks the SLID, DLID, SGID, DGID against the alternate path state). If the validation passes the QP transitions to the MIGRATED state.

Upon entry to the MIGRATED state, the primary path used by the QP logic for setting the outbound path and validating the inbound path are loaded with the alternate path state. At this point all request and response packets from both CAs are using the alternate path.

17.2.8.1.4 RE-ENABLING MIGRATION

Migration is re-enabled via management intervention. First the alternate path variables are reloaded with new alternate paths. Then, based on a command from a management entity, the QP state is set to REARM. This causes the MigReq bit in outbound packets to be set false. Upon receiving an inbound packet with MigReq set false, the QP state is set to ARMED. Migration at this point is now re-enabled.

17.3 HOST CHANNEL ADAPTER

A HCA is differentiated from a TCA in that it supports the architecturally defined IBA Verbs Layer. As such, an HCA (and its vendor specific and OS specific driver SW) shall support the functionality of the Verbs Layer chapter.

17.3.1 LOOPBACK

An HCA shall be able to internally loopback a packet sent to itself. That is, the verbs layer can specify a packet to be delivered to the same port (possibly a different QP though). The packet shall be delivered without the packet appearing on the port's physical link. This loopback shall be able to function without requiring the presence of an external switch.

InfiniBand does not reserve a special LID value to indicate loopback. Instead, the DLID (and DGID if present) of a loopback packet should be the LID (and GID) of the port on which the packet was emitted. For loopback packets, a channel adapter implementation may ignore other path information, such as MTU, that is not otherwise needed for the receive buffer or for the completion queue as specified in section 11.4.2.1 Poll for Completion on page 623.

On an HCA with multiple ports, a packet may be sent onto the wire from one port with the DLID in the packet targeting a different port. This is not

considered loopback and follows all the normal rules for sending packets. An external switch is required for such a packet transfer, there is no requirement that a packet be routed internally from one port to another.

Loopback packets for diagnostic purposes that traverse an external switch are performed by using the directed routed subnet management packets.

17.4 TARGET CHANNEL ADAPTER

A channel adapter that attaches an I/O node to the fabric is a Target Channel Adapter (TCA). In most regards, a TCA is indistinguishable from an HCA when viewed from the perspective of the IBA wire semantics. However, there are certain characteristics and requirements that distinguish a TCA from an HCA. This section describes some of the differences between a target channel adapter and a host channel adapter, specifies functionality required of a TCA, and specifies minimum requirements on a TCA.

This section also describes the role of the target channel adapter in supporting its clients. Figure 223 illustrates the relationship of the target channel adapter in an I/O node. The client of the target channel adapter's services is one or more I/O controllers.

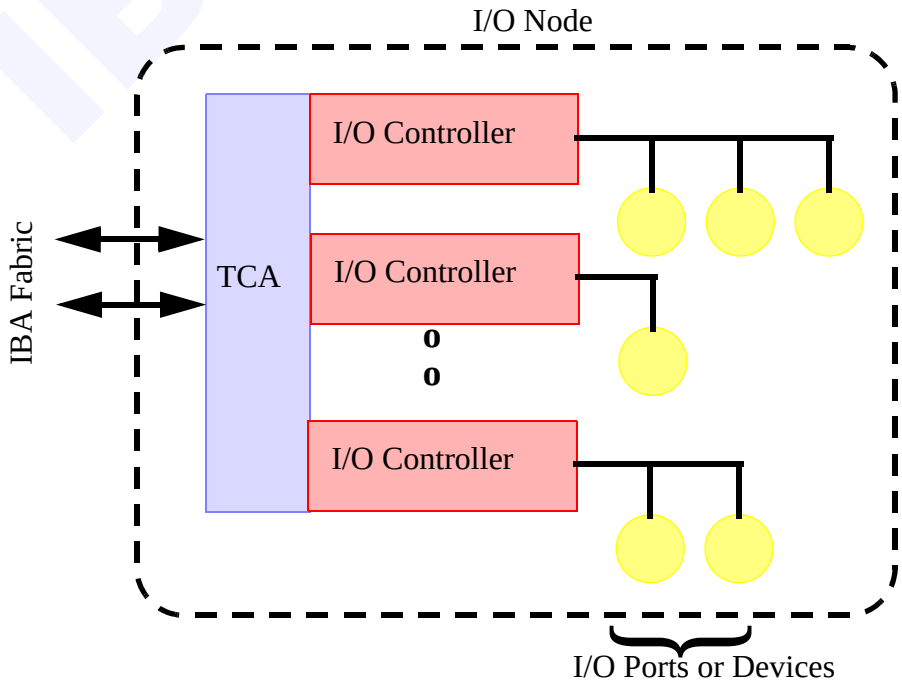


Figure 223 Generic I/O Node Model

17.4.1 CONTRAST TO A HOST CHANNEL ADAPTER

Unlike a host node, the execution environment for an I/O node is not necessarily associated with a general purpose processor. In fact, it can be entirely in hardware without any software environment.

For a host channel adapter, IBA specifies the semantics of the client interface characteristics (i.e., verbs) in order to support run time binding with the host's operating system and allow each component (HCA, OS, application) to be architected and distributed independently. But a target channel adapter can be bound to the I/O controller as part of the design process and distributed together. Thus the architecture does not specify any particular relationship between the target channel adapter and the I/O controller. This freedom promotes diversity and the ability to employ any queuing and notification mechanism that best serves the I/O function.

In the host environment, as illustrated, IBA service is separated into layers with the HCA hardware and the HCA driver being referred to as the host channel adapter. Thus the IBA services are not included in the requirements for the HCA. Instead, requirements for IBA services are applied to the host platform in general and not the HCA vendor.

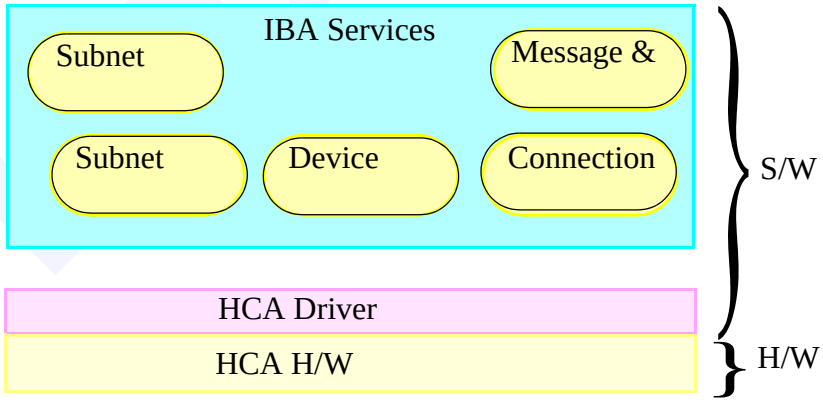


Figure 224 Host Environment - Split Responsibility

In the target environment, as illustrated, IBA services are not separated from TCA channel functionality, and thus the target channel adapter includes the IBA services. Thus the term target channel adapter is abstracted to mean all of the IBA mechanisms in the I/O node (target). TCAs may be implemented using software and hardware or hardware alone.

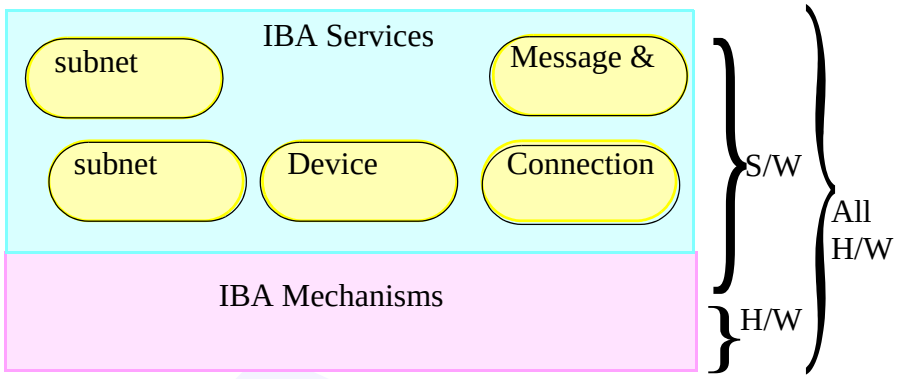


Figure 225 Target Environment - TCA Responsibility

A host channel adapter provides a generic service to its application "clients". Therefore, IBA requires that an HCA provide full channel functionality. This is because the HCA vendor does not have prior knowledge of what applications will run over its channels.

However, since a target channel adapter vendor may have prior knowledge of the way the target channel adapter will be applied, the hardware vendor can reasonably restrict a TCA's capabilities to only what is necessary for its clients.

17.4.1.1 MEMORY PROTECTION

IBA does not require that a TCA make any of its memory, or the memory associated with its attached I/O controllers directly accessible to an another channel adapter. That is, there is no requirement that a TCA be capable of accepting inbound Atomic or RDMA READ or WRITE requests. If the TCA does expose its memory, or that of its attached I/O controllers, the architecture does not require that the TCA provide any form of memory protection, nor prescribe any particular mechanism for registering or protecting access to that memory other than the mechanism provided in the transport layer.

17.4.2 DEVICE ADMINISTRATION

Device administration packets allow an I/O node's resources to be discovered and managed. In particular they provide the ability for a host to discover and invoke I/O services provided by I/O nodes.

A key distinction between a host and an I/O node lies in the method by which the I/O node's resources and capabilities are discovered, and the method by which connections to the TCA, and hence to the I/O resources on the node, are established. A complete set of messages is defined in Section 16.3 Device Management on page 985 for the purpose of allowing a consumer of the I/O node's services to discover the range of services offered by the I/O node.

Since a TCA is not required to support all IBA transport services, a particular TCA has associated with it a set of attributes defining its capabilities and the services it supports. Normally, these attributes are discovered by negotiation between peer channel adapters during the process of establishing a connection.

In the case of an I/O node which does not necessarily have the compute power and resources to participate in a complex negotiation, IBA defines a simple method by which a host or other intelligent I/O node can discover the target's attributes and establish connections accordingly.

Thus, IBA defines a rich set of I/O node attributes that can be read by an intelligent channel adapter and used during connection establishment in order to free the target from complex connection negotiation protocols. The host discovers target attributes directly, thus avoiding negotiation during connection establishment.

Each target may support the set of target/IO device attribute discovery messages as defined in Section 16.3.3 Attributes on page 989.

IBA does not specify the semantics nor methods between a TCA and its clients for conveying device information but it does specify the mechanisms and encoding for conveying that information.

IBA does not specify the semantics nor queueing models for the I/O controller to post messages, receive messages, and invoke RDMA Read, Write, and Atomic operations between a TCA and its clients.

IBA is I/O protocol agnostic. That is, how an I/O controller chooses to apply the services provided by the TCA is outside the scope of IBA as long as the usage corresponds to the rules for class of service and quality of service.

17.4.3 FABRIC LOOPBACK

A TCA does not have the same internal loopback requirement as does the HCA. Being a special purpose device, how the TCA handles packets addressed to itself is an implementation specific decision.

Loopback for diagnostic purposes that traverses an external switch is performed by using directed routed subnet management packets (just as done by the HCA)

CHAPTER 18: SWITCHES

18.1 OVERVIEW

This chapter specifies the requirements related to IBA switches.

Packets may be forwarded within a subnet (intra-subnet) and between subnets (inter-subnet). IBA switches are the fundamental forwarding component for intra-subnet routing (inter-subnet routing is provided by IBA routers, described later in this specification). Switches interconnect links by forwarding packets between the links.

Switches are transparent to the end stations and are not directly addressed (except for subnet management operations). To this end, every destination port within the network is configured with one or more unique Local Identifiers (LID's). From the point of view of a switch, a LID represents a path from the input port through the switch. Switch elements are configured with forwarding tables. Packets are addressed to their ultimate destination on the subnet using a destination LID (DLID), not to intervening switches. Individual packets are forwarded within a switch to an outbound port or ports based on the packet's DLID field and the Switch's forwarding table.

IBA switches are required to support unicast forwarding and may support multicast forwarding. In addition, IBA switches support a form of source routing, referred to as Directed Routing, for forwarding subnet management packets. This enables the configuration of a subnet without valid forwarding entries in the switches (e.g. a subnet power-up).

A Subnet Manager (SM) configures switches including loading their forwarding tables. The entity that communicates with the SM for the purpose of configuring the switch is referred to as the Subnet Management Agent (SMA). Every switch is required to have a subnet management agent. Individual switches within a power domain can be made observable to the SM via multiple instantiation of SMAs. Likewise, an SMA can be constructed that configures multiple switches and exports the multiple switches to the SM as a single switch; however, from the SM's perspective, such a configuration is a single switch.

Switches must also support a Subnet Management Interface (SMI) as specified in Chapter 14: Subnet Management on page 794 and a General Services Interface (GSI) as specified by Chapter 16: General Services on page 930. There are various mandatory and optional requirements of these interfaces that are specified in the respective chapters.

18.1.1 SWITCH PORT 0

IBA switches, in addition to external (or physical or data) ports, numbered 1 up to 254, have a special port which is numbered port 0 and also sometimes referred to as the management port. It is a unique port type in InfiniBand, being different from other IB ports (switch external ports, CA ports, or router ports being the other port types). There are now two subtypes of switch port 0 which are realizable by the IB architecture.

In addition to the base switch port 0, there is an optional enhanced switch port 0. Enhanced switch port 0 provides an additional architectural alternative to adding a switch port and a CA, either integrated or as separate components. Additionally, base switch port 0 is now more rigidly specified.

Switch port 0 is a virtual port as it is not required to be physically instantiated. It is exempted from compliance with the physical layer requirements specified in Chapter 6: Physical Layer Interface on page 163, but is required to be compliant with the link layer requirements specified in Chapter 7: Link Layer on page 167. Port 0 adheres to all IBA switch port requirements specified in Chapter 18: Switches on page 1040 with the exception that it may deviate from these requirements in any combination of the following ways:

- Port 0 is not required to be physically instantiated.
- Port 0 is not required to implement the IB physical layer electrical, optical, or mechanical requirements.
- Enhanced port 0 link speed will be set to the nearest InfiniBand equivalent, e.g. A PCI link may have the Linkwidth set to 1x.
- Enhanced port 0 PortPhysicalState shall support Linkup only.
- Port 0 is not required to implement IB link level flow control.

Further requirements for Port 0:-

- Port 0 is either in enhanced or base mode, an enhanced port can’t be reduced to a base port dynamically.

It also complies with the requirements of Chapter 9: Transport Layer on page 230 related to unreliable datagram service.

Enhanced switch port 0 is indicated by the EnhancedPort0 component in SwitchInfo SM attribute.

The direction of data flow to and from enhanced switch port 0, is viewed in the switch frame of reference. Data received by enhanced port 0 is controlled by the transmit components of PortInfo, and data transimitted from enhanced port 0 is controlled by the receive components of PortInfo.

18.2 DETAILED FUNCTIONAL REQUIREMENTS

18.2.1 ATTRIBUTES

This section describes the major architecturally defined attributes of switches that are left as implementation choices.

Unicast Forwarding Table:

C18-1: For the forwarding of unicast packets, a switch shall implement either a linear forwarding table or a random forwarding table, but not both.

C18-2: A switch shall implement a unicast forwarding table with at least one entry and no more than 49,152 entries.

Two forms of an unicast forwarding table are defined: linear and random. All switches support one and only one of these forwarding table types. In either case, the required size for the unicast forwarding table is not specified by IBA and may vary between implementations. However, a valid range of table sizes is specified. Switches that implement the random form may also choose to limit the number of entries that may be assigned to a given port. This is further described in section 18.2.4.3 Packet Relay on page 1048.

Multicast Support:

o18-1: The replication of multicast packets to multiple ports by switches is optional.

o18-2: A switch that implements the switch multicast replication service shall implement a multicast forwarding table with at least one entry and no more than 16383 entries.

IBA defines a switch multicast service that provides for the replication of packets by switches and their subsequent forwarding to multiple ports. The implementation of this service is optional. If implemented, IBA does not specify the size for the multicast forwarding table, and therefore the number of multicast groups a switch is capable of supporting. Consequently, the size of this table may vary by implementation. However, a valid range of table sizes is specified. Additional multicast requirements are specified in section 18.2.4.3.4 Optional Multicast Relay on page 1054.

Virtual lanes:

C18-3: Switches shall implement the subnet virtual lane (also referred to as virtual lane 15).

o18-3: Switches may implement a single buffer resource shared by all ports for the subnet management virtual lane.

All switches implement the subnet management virtual lane (which is numbered virtual lane 15). Additionally, switches implement one, two, four, eight, or 15 data virtual lanes. These virtual lanes are numbered sequentially starting with zero. Unlike data virtual lanes, buffering for virtual lane 15 may be shared by all ports and may be shared by packet reception and transmission. This is described in 7.6 Virtual Lanes Mechanisms on page 180.

SL to VL mapping:

o18-4: Switches that implement more than one data virtual lane shall implement the SL to VL mapping function specified in this chapter.

o18-5: Switches that implement one data virtual lane may implement the SL to VL mapping function specified in this chapter.

SL to VL mapping is required on switches that support more than one virtual lane in addition to virtual lane 15. It is optional on switches that support only one virtual lane in addition to virtual lane 15. The specific requirements of this table are described in section 7.6.6 VL Mapping Within a Subnet on page 186.

P_Key Enforcement:

o18-6: Switches may implement the Inbound P_Key Enforcement Service specified in this chapter.

o18-7: Switches may implement the Outbound P_Key Enforcement Service specified in this chapter.

Switches may enforce partitions on ingress to and/or egress from the switch. This mechanism is described in sections 18.2.4.2.1 Inbound P_Key Enforcement on page 1046 and 18.2.4.4.1 Outbound P_Key Enforcement on page 1054.

Maximum Transfer Unit (MTU) size:

C18-4: Switches shall be capable of forwarding LID routed packets of size from the minimum valid packet up to 382 bytes on the management virtual lane.

C18-5: Switches shall support one of the MTU sizes specified in Table 19 Packet Size on page 195 across all ports on the switch.

C18-6: This compliance statement is obsolete and has been replaced by C18-6.1.1:.

C18-6.1.1: With the exception of packets arriving on the management virtual lane, switches shall be capable of forwarding packets of size from the

minimum valid packet (Raw packets may be excluded) up to the supported MTU plus 126 bytes.

Table 19 Packet Size on page 195 specifies a choice of MTU that may be supported by IBA devices. Switch implementations support one of the specified MTU sizes for the entire switch. Switches are capable of forwarding packets whose size varies up to the maximum size indicated in the table for the implemented MTU size plus an additional 126 bytes.

Link Physicals:

IBA specifies various physical layer options. Switches may implement any of these options on any port and there is no requirement that all ports of a switch implement nor operate with the same physical options. Switches conform to the detailed requirements for physical layer support as specified in Chapter 6: Physical Layer Interface on page 163.

18.2.2 INITIALIZATION

C18-7: Upon power-up, a switch shall be initialized to the following state:

- All initialization of attributes as required in Chapter 14: Subnet Management on page 794.
- Physical and link layers shall be reset.
- All virtual lane queues shall be cleared.
- P_Key enforcement, if implemented, shall be disabled for all ports.
- The NeighborMTU component of each PortInfo attribute shall be initialized to indicate 256 byte MTU as specified in 14.2.5.6 PortInfo on page 821.

Note that a switch contains many tables, some of which are optional. These include the forwarding table, the SL to VL mapping table, the multicast forwarding table, P_Key tables, etc. There is no requirement for a switch to initialize any of these tables; the subnet manager is responsible for appropriate initialization.

18.2.3 CONFIGURATION

Switches are configured via a subnet manager. Switches support the required subnet management operations and may support the optional subnet management operations specified in Chapter 14: Subnet Management on page 794.

18.2.4 PACKET RELAY REQUIREMENTS

The primary function of IBA switches is the relay of packets between links. This section specifies the requirements for supporting this function. This section assumes normal operation; required operation under error conditions is specified in section 18.2.5 Error Handling on page 1056.

To simplify the explanation of switch requirements, this section is divided into several architectural functions. This division does not imply a particular implementation; it is done solely to enhance the organization of the specification.

18.2.4.1 SWITCH PORTS

C18-8: Each port on an IBA Switch except port 0 shall comply with the physical layer requirements specified in Chapter 6: Physical Layer Interface on page 163.

C18-9: Each physically instantiated external port on an IBA Switch shall comply with the link layer requirements specified in Chapter 7: Link Layer on page 167 of this specification.

C18-10: Base switch port number 0 shall, at a minimum, support the forwarding of packets to and from the switch's Subnet Management Interface and General Services Interface.

C18-10.a1: A switch may support up to 254 physical ports. For a switch supporting N physical ports, the ports shall be numbered from 1 to N.

C18-11: Port number 0 shall comply with the requirements of Chapter 9: Transport Layer on page 230 related to unreliable datagram service.

o18-8: Port 0 shall adhere to all IBA switch port requirements specified in this chapter with the exception that it may deviate from these requirements in any combination of the following ways:

- Port 0 is not required to be physically instantiated.
- Port 0 is not required to implement the IB physical layer electrical, optical, or mechanical requirements.
- Port 0 is not required to implement IB link level flow control.

C18-12: This compliance statement is obsolete.

C18-13: This compliance statement is obsolete.

C18-14: This compliance statement is obsolete.

Base switch port 0 is assigned a LID similar to that of channel adapters; however, unlike channel adapters, this port does not support multipathing and an LMC value cannot be assigned. The LID is assigned using the LID component of the PortInfo attribute. Refer to 14.2.5.6 PortInfo on page 821 for details on these requirements.

C18-14.a1: Enhanced switch port 0 shall comply with TCA compliances in chapter 17 with the exception of the following:

- Physical interfaces
- Port Numbering

18.2.4.2 RECEIVER QUEUING

The receiver queueing function receives packets from the link layer defined in Chapter 7: Link Layer on page 167.

C18-15: The virtual lane into which an individual packet is queued shall be the one corresponding to the VL field in the packet's Local Route Header.

C18-16: This compliance statement is obsolete.

o18-8.a1: If the FilterRawInbound component of the receiving port's PortInfo Attribute is set to one, then the switch shall discard all packets received on that port in which the LNH field of the LRH contains binary 00 or binary 01 (i.e. raw packets).

C18-17: Switches shall not discard packets in lieu of implementation of the link level flow control as specified in section 7.9 Flow Control on page 209.

18.2.4.2.1 INBOUND P_KEY ENFORCEMENT

The implementation of the inbound P_Key enforcement service in switches is optional. This section specifies the requirements of the service if implemented.

Inbound P_Key verification is enabled and disabled for each port individually based on the PatrtitionEnforcementInbound component of the PortInfo attribute.

o18-9: If a switch provides the inbound P_Key enforcement service and the PartitionEnforcementInbound component of the PortInfo Attribute is set to zero, then the inbound P_key enforcement service shall be disabled for packets received on the corresponding port.

o18-10: If a switch provides the inbound P_Key enforcement service, it shall maintain a separate list of P_Keys associated with each port.

o18-11: If a switch provides both the inbound P_Key enforcement service and the outbound P_Key enforcement service, then the list of P_Keys associated with each port shall be the same list for both the inbound P_Key enforcement service and the outbound P_Key enforcement service.

o18-12: If a switch provides the inbound P_Key enforcement service, the P_Key table associated with each port shall be capable of containing be-

tween one and 65535 P_Keys, inclusive (the exact number is left as an implementation parameter).

o18-13: If a switch provides the inbound P_Key enforcement service, the P_Key table associated with each port shall be programmable using the P_KeyTable attribute defined in 14.2.5.7 P_KeyTable on page 834.

o18-14: If a switch provides the inbound P_Key enforcement service and if the PartitionEnfocementInbound component of the PortInfo Attribute is set to one, then any packet received on a virtual lane other than 15 shall either be discarded or truncated such that it contains no data past the BTH if the value in the P_Key field in the BTH does not match one of the entries in the receiving port's P_Key list and either of the following conditions are true:

- LNH field in the LRH contains binary 11 and IPVer field in the GRH contains 6.
- LNH field in the LRH contains binary 10.

For the purpose of inbound P_Key enforcement, a P_Key matches an entry in the P_Key table if and only if it is not the invalid P_Key one of the following conditions are true:

- The P_Key membership bit in the packet is full and there is an entry in the P_Key table that equals all 16 bits of the P_Key
- The P_Key membership bit in the packet is limited and there is an entry in the P_Key table whose 15 bits exclusive of the membership bit equal those bits in the P_Key.

o18-15: If a switch provides the inbound P_Key enforcement service and if the PartitionEnfocementInbound component of the PortInfo Attribute is set to one, then any packet received on a virtual lane other than 15 shall either be discarded or truncated in length such that it contains no more than 64 bytes if all of the following conditions are true:

- LNH field of the LRH contains binary 11.
- IPVer field of the GRH does not contain 6.

o18-16: If a switch provides the inbound P_Key enforcement service and if the PartitionEnforcementInbound component of the PortInfo Attribute is set to one, then any packet that is too short to contain a BTH and that the LNH field contains binary 11 shall be discarded or shall be forwarded with the EBP delimiter appended and with the inverse of the valid VCRC.

Raw packets, i.e. packets in which the LNH field of the LRH contains binary 00 or binary 01, are not subject to P_Key enforcement and are not discarded nor truncated by this mechanism.

18.2.4.3 PACKET RELAY

Packet relay refers to the operation of transferring a packet from the virtual lane on the inbound port to the virtual lane on one or more a outbound ports.

C18-18: A switch shall relay each unicast packet from the data virtual lane(s) in which it was received to the output port indicated by the unicast forwarding table entry corresponding to the packet's DLID field.

A switch performs this relay function regardless of the state of the destination port. In certain states, the destination port will discard the packet. This is described in detail in 7.2 Link States on page 168.

C18-19: Each packet received on virtual lane 15 in switches that implement independent buffering for virtual lane 15 on each port shall be relayed to the virtual lane 15 on the output port indicated by the unicast forwarding table entry corresponding to the packet's DLID field.

C18-20: If a LID routed packet is relayed to the same port on which it was received, it shall be discarded.

(Note: Directed route packets are permitted to be transmitted from the port from which they were received. This does not violate the above requirement since the packet is actually relayed from the received port to port 0, the SMI; then it is received from port 0 and transmitted out the original port).

C18-21: No packet contents shall be modified by the switch except as required by this specification.

This chapter specifies various conditions under which a packet may or must be truncated in length. These conditions do not imply that the PktLen or PayLen fields may be modified.

C18-22: Packets received on ports other than port 0 with a DLID equal to the permissive address shall be forwarded to port 0.

(Note: A switch performs this forwarding of packets received with the permissive DLID regardless of whether it implements a linear forwarding table or a random forwarding table for unicast packets.)

o18-17: If base switch port 0, the SMI, or GSI of a switch does not contain sufficient free buffering to receive the packet, the packet may be discarded.

A special address, the permissive address (see 4.1 Terminology And Concepts on page 142) is defined by IB to permit the subnet manager to com-

municate with the SMI without knowledge of the LID assigned to the SMI. Packets with the permissive address received on ports other than 0 are always forwarded to port 0.

C18-23: Packets with the permissive address received on port 0 (i.e. generated by the SMI) shall be forwarded to the port specified by the SMI.

The mechanism for the SMI to specify the port is not defined by IB and may vary by implementation.

o18-18: Switches that support more than one virtual lane in addition to the management virtual lane (virtual lane 15), shall set the value of the VL field in the local route header as defined in section 7.6.6 VL Mapping Within a Subnet on page 186.

o18-19: Switches that support one virtual lane in addition to the management virtual lane (virtual lane 15), may implement VL Mapping as defined in section 7.6.6 VL Mapping Within a Subnet on page 186.

o18-20: Switches that support more than one virtual lane in addition to the management virtual lane (virtual lane 15), shall relay packets to the VL of the output port as defined in section 7.6.6 VL Mapping Within a Subnet on page 186 if the corresponding VL is implemented on the output port.

o18-21: Switches that support more than one virtual lane in addition to the management virtual lane (virtual lane 15), shall discard packets if the output VL as defined in section 7.6.6 VL Mapping Within a Subnet on page 186 is not implemented on the output port.

o18-22: Switches that support only one virtual lane in addition to the management virtual lane (virtual lane 15) shall not modify the VL field.

C18-24: Switches that support only one virtual lane in addition to the management virtual lane (virtual lane 15) shall relay packets to the VL of the output port indicated by the VL field in the LRH.

For switches that support only one data virtual lane, the link layer will discard all packets that do not contain either 0 or 15 in the VL field, therefore, there is no need for the relay function to modify the VL field in this case.

C18-25: Except for virtual lane 15, if the virtual lane on the outbound port does not contain sufficient space for the packet to be relayed, then the packet shall remain in the virtual lane on the inbound port until sufficient space is available or until the switch lifetime limit mechanism permits the discard of the packet.

C18-26: If the relay function is unable to relay packet from an inbound port to an outbound port due to lack of sufficient space in the outbound VL, the

relay function shall continue to relay packets from other virtual lanes destined for virtual lanes on outbound ports with sufficient space.

o18-23: In switches that implement independent buffering on each port for virtual lane 15, if when relaying virtual lane 15 packets the virtual lane on the output port does not contain sufficient space for the packet to be relayed, then the packet may be discarded.

C18-27: Packets shall be transmitted on a given port and SL in the same order as they were received from a given port except that ordering between unicast and multicast packets is not required.

o18-24: The relay function may, but is not required to, relay packets in the inbound portion of virtual lanes that are behind packets that are blocked due to insufficient space in the outbound portion of virtual lanes.

The method of arbitration when multiple inbound VLs have packets destined for the same outbound VL is left to the implementor, but the arbitration should service all inbound ports fairly.

C18-28: The forwarding table shall be configured in one of two ways, linear or random, as defined in section 18.2.4.3.1 Linear Forwarding Table Requirements on page 1050 and 18.2.4.3.2 Random Forwarding Table Requirements on page 1051.

C18-29: Switches shall conform to the requirements in section 18.2.4.3.3 Required Multicast Relay on page 1053.

o18-25: Switches may implement the requirements in section 18.2.4.3.4 Optional Multicast Relay on page 1054.

C18-30: A switch that does not implement the optional multicast relay shall set the MulticastFDBCap component of the SwitchInfo attribute to zero.

18.2.4.3.1 LINEAR FORWARDING TABLE REQUIREMENTS

This section describes the requirements related to the linear forwarding table. The linear forwarding table provides a simple map from LID to destination port. Conceptually, the table itself contains only destination ports; the LID acts as an index into the table from which the packet's destination address is obtained.

C18-31: In switches that implement the linear forwarding table, the linear forwarding table shall contain a port entry for each LID starting from zero and incrementing by one up to the size of the forwarding table.

C18-32: In switches that support the linear forwarding table, the size of the linear forwarding table shall be advertised in the LinearFDBCap component of the SwitchInfo attribute.

C18-33: In switches that support the linear forwarding table, the RandomFDBCap component of the SwitchInfo attribute shall be set to zero.

C18-34: In switches that implement the linear forwarding table, the linear forwarding table shall be programmable using the LinearForwardingTable attribute as described in 14.2.5.10 LinearForwardingTable on page 837.

Note that forwarding to switch port 0 (including the SMI/GSI) is enabled by programming the corresponding entries in the forwarding table to port 0. Setting the LID component of the PortInfo attribute does not automatically load this value in the forwarding table.

C18-35: A switch that implements a linear forwarding table shall support the SM programmable LinearFDBTop component of the SwitchInfo attribute as described in 14.2.5.4 SwitchInfo on page 819.

C18-36: Switches that implement linear forwarding tables shall discard all unicast packets that meet any of the following conditions:

- the packet's DLID value is greater than the value of LinearFDBTop and is not the permissive address
- the packet's DLID is outside the range supported by the linear forwarding table and is not the permissive address
- the port number in the forwarding table corresponding to the packet's DLID is set to a port that does not exist.

18.2.4.3.2 RANDOM FORWARDING TABLE REQUIREMENTS

This section describes the requirements related to the random forwarding table. Conceptually, the random forwarding table acts as a "content addressable memory"; it is loaded with both LIDs and destination ports. The table is "addressed" by a packet's LID and the corresponding destination port is returned. A switch implementation can limit the number of LIDs that correspond to a given port to as few as one. This enables the implementation of a "leaf" switch, i.e., a switch that supports only the connection of CA's to all ports but one. Such a switch requires a very small forwarding table (one LID per port). Such limitations are neither mandated nor prohibited by this specification.

C18-37: In switches that implement the random forwarding table, the random forwarding table shall provide for the storage of a set of unicast LID/LMC pairs and corresponding destination port entries.

C18-38: Switches that implement the random forwarding table shall maintain a DefaultPort value which shall be programmable via the DefaultPort component of the SwitchInfo attribute (see 14.2.5.4 SwitchInfo on page 819 for additional detail).

C18-39: Packets that arrive on ports other than the port indicated by DefaultPort with a unicast DLID field that does not match an entry in the random forwarding table and is not equal to the permissive address shall be forwarded to the port indicated by DefaultPort.

C18-40: If the DefaultPort value is a port that does not exist then packets that would otherwise be forwarded to this port shall be discarded.

C18-41: Packets that arrive on the port indicated by DefaultPort with a unicast DLID field that is not the permissive address and does not match an entry in the random forwarding table shall be discarded.

Matching an entry in the table means that the packet's DLID matches the LID in the table excluding the LMC least significant bits.

C18-42: Switches that implement the random forwarding table shall advertise the size of the table, i.e. the number of LID/LMC pairs that it may contain, in the RandomFDBCap component of the SwitchInfo attribute.

C18-43: Switches that implement the random forwarding table shall set the LinearFDBCap component of the SwitchInfo attribute to zero.

o18-26: Switches that implement a random forwarding table may limit the number of LID/LMC pairs that can be assigned to a given port.

C18-44: If a switch that implements the random forwarding table limits the number of LID/LMC pairs that can be assigned to a given port, then it shall set the LIDsPerPort component of the SwitchInfo component to the number of LIDs that is supported per port.

C18-45: If a switch that implements the random forwarding table does not impose such limitation on the number of LID/LMC pairs that can be assigned to a given port, it shall set the value of the LIDsPerPort component the same as the RandomFDBCap component.

C18-46: In switches that support the random forwarding table, the random forwarding table shall support exactly one LID/LMC entry.

(Note: The preceding compliance statement implies that if a LID/LMC pair is loaded into multiple entries of the random forwarding table such that the LID/LMC is not uniquely associated with a single output port, the output port used for packets sent to this LID/LMC is not guaranteed.)

Note: Switches that impliment a Random Forwarding Table shall discard all unicast packets for which the port number in the forwarding table entry corresponding to the packet's DLID is set to a port identifier that does not exist.

Note that forwarding to switch port 0 (including the SMI/GSI) is enabled by programming an entry in the forwarding table to port 0. Setting the LID component of the PortInfo attribute does not automatically load this value in the table.

The LIDsPerPort component does not apply to port 0.

18.2.4.3.3 REQUIRED MULTICAST RELAY

C18-47: All switches shall maintain values for a default primary multicast port and a default non-primary multicast port.

All switches maintain values for default primary multicast port and a default non-primary multicast port regardless of whether the switch supports multicast forwarding and regardless of the type of unicast forwarding table implemented.

C18-48: Switches shall allow the SM to set the values of the default primary multicast port and a default non-primary multicast port using the DefaultMulticastPrimaryPort and DefaultMulticastNotPrimaryPort components of the SwitchInfo attribute.

C18-49: All multicast packets that are received on ports other than the default multicast primary port shall be forwarded to the default multicast primary port if any of the following conditions are true:

- The switch does not implement a multicast forwarding table.
- The switch implements a multicast forwarding table and the multicast DLID in the packet is outside the range of the multicast forwarding table.
- The switch implements a multicast forwarding table and the entry in the forwarding table corresponding to the packet's DLID is zero.

C18-50: All multicast packets that are received on the default multicast primary port shall be forwarded to the default multicast non-primary port if any of the following conditions are true:

- The switch does not implement a multicast forwarding table.
- The switch implements a multicast forwarding table and the multicast DLID in the packet is outside the range of the multicast forwarding table.

- The switch implements a multicast forwarding table and the entry in the forwarding table corresponding to the packet's DLID is zero.

C18-51: If either the default multicast primary port or default multicast non-primary port is set to a port that does not exist then multicast packets that would otherwise be forwarded to the corresponding port shall be discarded.

18.2.4.3.4 OPTIONAL MULTICAST RELAY

This section describes the requirements for the optional replication of multicast packets.

o18-27: The replication of packets as part of multicast relay is optional.

o18-28: Switches that support multicast packet replication shall implement a multicast forwarding table that contains a port entry for each multicast LID starting from 0xc000 and sequentially incrementing to include the total number of multicast entries supported.

o18-29: In switches that support multicast packet replication, the number of multicast entries supported in the multicast forwarding table shall be at least one and no greater than 16383.

o18-30: In switches that support multicast packet replication, the number of multicast entries supported in the multicast forwarding table shall be advertised in the MulticastFDBCap component of the SwitchInfo attribute.

o18-31: In switches that support multicast packet replication, if the DLID of a packet is a multicast LID, then the switch shall relay the packet to the set of ports, excluding the port on which the packet was received, indicated by the multicast forwarding table entry corresponding to the packet's DLID field.

A switch performs this relay function regardless of the state of the destination port. In certain states, the destination port will discard the packet. This is described in detail in 7.2 Link States on page 168.

o18-32: In switches that support multicast packet replication, the virtual lane field shall be updated in each replicated packet in the same manner as for unicast packets.

18.2.4.4 TRANSMITTER QUEUING

Relayed packets are queued in the outbound portion of virtual lanes.

18.2.4.4.1 OUTBOUND P_KEY ENFORCEMENT

The implementation of the outbound P_Key enforcement service in switches is optional. This section specifies the requirements of the service if implemented.

Outbound P_Key verification shall be enabled and disabled for each port individually based on the PatrtitionEnforcementOutbound component of the PortInfo attribute.

o18-33: If a switch provides the outbound P_Key enforcement service and the PartitionEnfocementOutbound component of the PortInfo Attribute is set to zero, then the outbound P_key enforcement service shall be disabled for packets received on the corresponding port.

o18-34: If a switch provides the outbound P_Key enforcement service, it shall maintain a separate list of P_Keys associated with each port.

o18-35: If a switch provides both the inbound P_Key enforcement service and the outbound P_Key enforcement service, then the list of P_Keys associated with each port shall be the same list for both the inbound P_Key enforcement service and the outbound P_Key enforcement service.

o18-36: If a switch provides the outbound P_Key enforcement service, the P_Key table associated with each port shall be capable of containing between one and 65535 P_Keys, inclusive (the exact number is left as an implementation parameter).

o18-37: If a switch provides the outbound P_Key enforcement service, the P_Key table associated with each port shall be programmable using the P_KeyTable attribute defined in 14.2.5.7 P_KeyTable on page 834.

o18-38: If a switch provides the outbound P_Key enforcement service and if the PartitionEnfocementOutbound component of the PortInfo Attribute is set to one, then any packet to be transmitted on a virtual lane other than 15 on that port shall either be discarded or truncated such that it contains no data past the BTH if the value in the P_Key field in the BTH does not match an entry in the transmitting port's P_Key list and either of the following conditions are true:

- LNH field in the LRH contains binary 11 and IPVer field in the GRH contains 6.
- LNH field in the LRH contains binary 10.

For the purpose of outbound P_Key enforcement, a P_Key matches an entry in the P_Key table if and only if the P_Key is not the invalid P_Key and one of the following conditions are true:

- The P_Key membership in the packet bit is limited and there is an entry in the P_Key table whose membership bit is full and whose remaining 15 bits equal those of the P_Key
- The P_Key membership bit in the packet is full and there is an entry in the P_Key table whose 15 bits exclusive of the membership bit equal those bits in the P_Key.

o18-39: If a switch provides the outbound P_Key enforcement service and if the PartitionEnforcementOutbound component of the PortInfo Attribute is set to one, then any packet to be transmitted on a virtual lane other than 15 of that port shall either be discarded or truncated in length such that it contains no more than 64 bytes if all of the following conditions are true:

- LNH field of the LRH contains binary 11.
- IPVer field of the GRH does not contain 6.

o18-40: If a switch provides the outbound P_Key enforcement service and if the PartitionEnforcementOutbound component of the PortInfo Attribute is set to one, then any packet that is too short to contain a BTH and that the LNH field contains binary 11 shall be discarded or shall be forwarded with the EBP delimiter appended and with the inverse of the valid VCRC.

Raw packets, i.e. packets in which the LNH field of the LRH contains binary 00 or binary 01, are not subject to P_Key enforcement and are not discarded nor truncated by this mechanism

18.2.4.5 PACKET TRANSMISSION

C18-52: This compliance statement is obsolete.

o18-40.a1: If the FilterRawOutbound component of the transmitting port's PortInfo attribute is set to one, then the switch shall discard all packets to be transmitted on that port in which the LNH field of the LRH contains binary 00 or binary 01 (i.e. raw packets).

C18-53: Each packet shall be transmitted with a valid VCRC field computed as specified in section 7.8.2 Variant CRC (VCRC) - 2 Bytes on page 197, unless required otherwise in this chapter or in chapter Chapter 7: Link Layer on page 167.

C18-54: Each packet shall be transmitted with an egp character appended unless required otherwise in this chapter.

C18-55: Switches shall support the requirements specified in 7.6.9 VL Arbitration and Prioritization on page 188.

18.2.5 ERROR HANDLING

This section specifies required operation under error conditions. Like the previous section, this section is divided into several architectural functions. This division does not imply a particular implementation; it is done solely to enhance the organization of the specification.

18.2.5.1 SWITCH PORTS

C18-56: This compliance statement is obsolete.

C18-57: This compliance statement is obsolete.

There are no additional switch port error handling requirements.

18.2.5.2 RECEIVER QUEUING

There are no additional receiver queuing error handling requirements.

18.2.5.3 PACKET RELAY

There are no additional packet relay error handling requirements.

18.2.5.4 TRANSMITTER QUEUEING

The transmitter packet discard is based on, among other things, two time values: Switch Lifetime Limit (SLL) and Head of Queue Lifetime Limit (HLL).

SLL is defined as 4.096us * 2^{LV} if $0 \leq LV \leq 19$, +5% / -55%. LV is the LifeTimeValue component of the SwitchInfo attribute. If LV > 19, then SLL is to be interpreted as infinite.

HLL is defined as 4.096us * 2^{HL} if $0 \leq HL \leq 19$, +5% / -55%. HL is the HOQLife component of the PortInfo attribute. If HL > 19, then HLL is to be interpreted as infinite.

C18-58: The transmitter queueing function shall discard any packet that meets any of the following conditions:

- The packet has been at the head of the Virtual Lane (i.e. the position to be transmitted next), and has not begun transmission within HLL.
- The packet is queued to a VL that is in the VL stalled state. If VLStallCount sequential packets are discarded from a given VL due to exceeding the HLL requirement above, the VL shall enter the VL stalled state. A VL shall leave the VL stalled state 8 * HLL after entering it. VLStallCount component is provided in the PortInfo attribute.
- The size of the packet as indicated by the PktLen field exceeds the MTU supported by the neighbor device as indicated by the NeighborMTU component of the PortInfo attribute.

C18-59: If a switch by virtue of its implementation cannot guarantee that any packet entering it will be transmitted within 2.5 ms, measured first bit in to first bit out and assuming flow control credit is continuously available, then it shall discard any packet that has not begun transmission within SLL measured from the time the first bit was received by the switch.

o18-41: If a switch by virtue of its implementation can guarantee that any packet entering it will be transmitted within 2.5 ms, measured first bit in to first bit out and assuming flow control credit is continuously available, then it may discard any packet that has not begun transmission within SLL measured from the time the first bit was received by the switch.

18.2.5.5 PACKET TRANSMISSION

C18-60: Each packet to be transmitted that is truncated in length as permitted or specified by any condition in this chapter be corrupted as specified in 7.3 Packet Receiver States on page 172.

18.2.6 SUBNET MANAGEMENT AGENT REQUIREMENTS

C18-61: Switches shall support a Subnet Management Interface (SMI) as specified in Chapter 14: Subnet Management on page 794.

C18-62: Switches shall support a General Services Interface (GSI) as specified by Chapter 16: General Services on page 930.

There are various mandatory and optional requirements of these interfaces that are specified in the respective chapters.

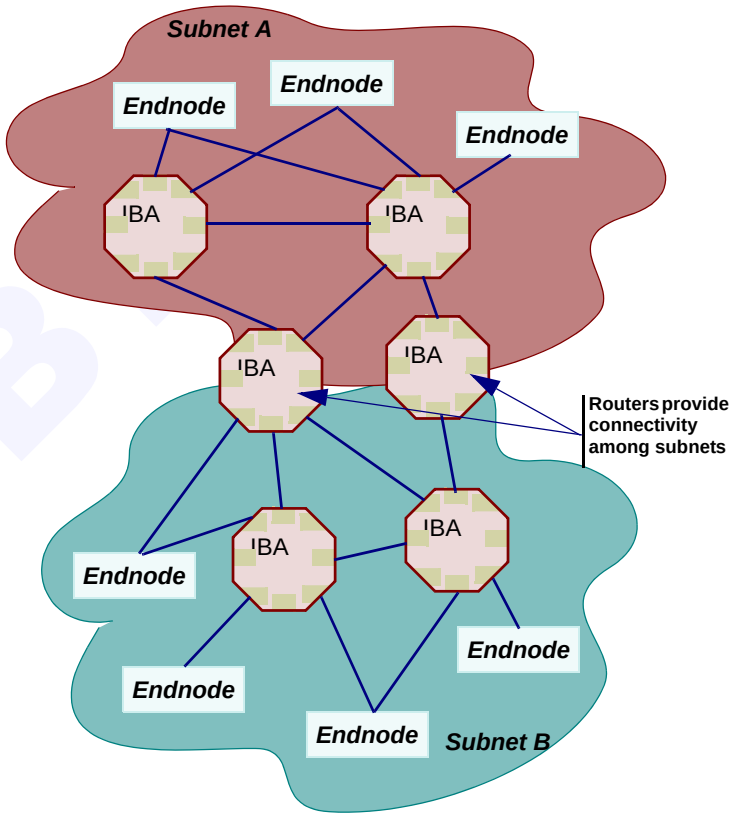
C18-63: Switches shall implement P_Key checking on the GSI as specified in section 10.9.8 Partition Enforcement on Management Queue Pairs on page 529.

CHAPTER 19: ROUTERS

19.1 OVERVIEW

IBA Routers are IBA packet relay devices, that operate at the network layer of the IBA addressing hierarchy to interconnect multiple locally addressed subnets. As the top level in the hierarchy, IBA Routers rely on global identifiers (GIDs).

Figure 226 Reference of Routers Connecting Subnets



IBA Router usage is meant to satisfy:

1) Scalability
2) Local address space reuse
3) Containment of failures and topology changes
4) Confinement of fabric management scope to subnets

In fulfilling these objectives, IBA Routers also allow the IBA semantics, and QoS characteristics to be extended across IBA subnets.

IBA Routers are required to support unicast routing and may support multicast routing. The specification of the routing forwarding mechanisms in this chapter is presently limited to unicast routing.

IBA Routers use destination based routing, where every destination port within the global fabric is assigned one or more unique Global Identifiers (GID). From the point of view of a Router, a GID represents either an endnode port or another router's port on a directly attached subnet. A GID does not necessarily represent a path through the fabric, as the Router is allowed to spread traffic over several paths based on other packet header criteria.

A Router is visible to IBA nodes on the directly attached subnets, and it is transparent to nodes on any remote subnet. The Subnet Manager address resolution function makes local routers visible to endnodes; endnodes in turn use this information when addressing packets to a local router LID on their way to a remote destination. Routers on the same subnet are also visible to each other, both for the purpose of implementing a routing protocol, and also when routing packets through other routers as the next hop. Finally, routers on a subnet are also visible to Subnet Managers on their respective directly attached subnets.

Each Router port must support a Subnet Management Interface (SMI) (see 13.5.1.1 Processing Subnet Management Packets (SMPs) on page 751) and a General Services Interface (GSI) (see 13.5.1.2 Processing General Services Management Packets (GMPs) on page 752). There are various mandatory and optional requirements of this interface that are specified in the management chapter. Subnet Managers assign LIDs to IBA Router ports and provide a service to find a path to other endnodes or routers. The Router Management section of the Management chapter defines the attributes of the interactions between Subnet Managers and Routers.

19.2 DETAILED FUNCTIONAL REQUIREMENTS

The present IBA Router specification does not cover the routing protocol nor the messages exchanged between routers. Future revisions of this chapter will complete such control functions.

19.2.1 ATTRIBUTES

IBA Routers reside at the boundaries between subnets, and are configured separately per port by different Subnet Managers and at different times. Subnet managers supply IBA Routers with LIDs/LMCs (for each

port separately), and additional path information like SL to VL mappings and MTU values.

Unicast Routing Table:

C19-1: A router shall implement a unicast routing table with at least as many entries as the number of router ports.

IBA Routers have routing tables for their active routes. These tables are hierarchical and include explicit endnode routes (e.g. last hop to endnode), prefix routes (aggregate route for entire subnet), and possibly default routes (routes for unknown prefixes). The size of the unicast routing table is implementation dependent.

Virtual lanes:

C19-2: Routers shall implement the subnet management virtual lane (also referred to as virtual lane 15).

C19-3: Router ports shall implement data virtual lanes as specified in 7.6 Virtual Lanes Mechanisms on page 180, in addition to the subnet management virtual lane, numbered virtual lane 15.

C19-4: Virtual lane 15 shall be implemented independently for each router port.

C19-5: Routers shall not route any VL15 packets between router ports.

SL to VL mapping:

C19-6: Routers that implement more than one data virtual lane shall implement the SL to VL mapping function specified in this chapter.

o19-1: Routers that implement one data virtual lane may implement the SL to VL mapping function specified in this chapter.

Per port SL to VL mapping is required on Routers that support more than one virtual lane in addition to virtual lane 15. It is optional on Routers that support only one virtual lane in addition to virtual lane 15.

Tclass to SL mapping:

C19-7: Routers shall preserve the Tclass value when routing.

SL values may be replaced when routing into a different subnet. The Tclass value is preserved, as it represents the Class of Service with end-to-end IBA scope. The Tclass to SL mapping function is not defined by the present specification revision.

P_Key Enforcement:

o19-2: Routers may implement the Inbound P_Key Enforcement Service specified in this chapter.

o19-3: Routers may implement the Outbound P_Key Enforcement Service specified in this chapter.

Routers may enforce partitions on ingress to and/or egress from the Router.

Maximum Transfer Unit (MTU) size:

C19-8: Each router port shall independently support one of the MTU sizes specified in Table 19 Packet Size on page 195.

C19-9: Routers shall be capable of routing packets of size from the minimum valid packet size up to the supported MTU of the intervening ports plus 126 bytes.

Table 19 Packet Size on page 195 specifies a choice of MTU that may be supported by IBA devices. Router implementations independently support one of the specified MTU sizes for each port. Packets exceeding the MTU size of the participating links may be discarded or truncated. Each port provides sufficient buffering for each data VL to advertise credit for at least one packet with MTU payload.

Link Physicals:

IBA specifies various physical layer options. Routers may implement any of these options on any port and there is no requirement that all ports of a Router implement nor operate with the same physical options. Routers shall conform to the detailed requirements for physical layer support as specified in Chapter 6: Physical Layer Interface on page 163.

End-to-end data integrity:

Although IBA routers modify some packet headers during routing, none of these headers affects the value of the ICRC, and IBA Routers shall preserve the original ICRC rather than recomputing its value locally.

19.2.2 INITIALIZATION

C19-10: Upon power-up, a router shall be initialized to the following state:

- All initialization defined in Chapter 13: Management Model on page 709 and applicable to routers.
- Physical and link layers reset.

- All virtual lane queues shall be cleared.
- Other routing table entries and all queues (and any other table type structures) shall be cleared.

19.2.3 CONFIGURATION

Routing tables - Entries may be derived from any combination of externally configured routes and autonomously computed routes. In particular, routers rely on the SM database for routes to endnodes on directly attached subnets.

SL mapping tables - SL to VL mapping tables exist at every router port.

Tclass mapping - Any necessary configuration of the Tclass end-to-end class of service role in determining the local SL value is configured into the router.

19.2.4 PACKET RELAY MODEL

The logical abstraction for IBA packet routing is that of packet by packet routing and is given by:

if

i) ((BASE DLID == router port BASE LID) AND
ii) (LRH:Next Header == GRH) AND
iii) (Destination GID <> router GID) AND
iv) (Destination GID matches entry in route table) AND
v) (VCRC OK) AND (Hop count > 1))

then {

i) Replace DLID with value from routing table
ii) Replace SLID with LID of output port
iii) Replace SL, considering Tclass (among other possible criteria)
iv) Map SL to VL using per output port table
v) Decrement Hop count
vi) Recompute VCRC, preserve ICRC

}

Note: Routers may also check ICRC, or just rely on the end-to-protection of endnodes checking ICRC.

The above abstraction, combined with the Network layer Addressing model, dictates a longest match against the destination GID. Implementa-

tions may exploit the addressing model to relax this function to a combination of a 64-bit longest match for prefix type entries, and either a 64-bit or 128-bit fixed length match for explicit routes, depending on the uniqueness scope of the lowest 64 bits of the GID.

19.2.4.1 PATH SELECTION

A Router may support multiple paths to a given DGID. These may include paths via the same next-hop and/or different next-hops as well as different paths within the subnet (LMC based) to a given GID.

An IBA Router may actively use multiple paths with equal or different costs, as long as it does not affect ordering by separating packets of a given session.To allow IBA Routers to have different degrees of sophistication in determining what packets may be separated, a session is used in a deliberately vague way.

The baseline assumption is that endpoints will use identical GRH:FlowLabel values for sequences of packets whose relative ordering is important, therefore a possible session representation at the router would be the **(DGID, SGID, TClass, SL, FlowLabel)** tuple. A router may use other attributes to select a path but once selected that path will continue to be used for subsequent packets unless a management event dictates a new path.

19.2.4.2 ROUTER PORTS

A router with N physical ports, associates PORTINFO attributes to its physical ports based on the Port Number Attribute Modifier value, ranging between 1 and N.

C19-11: Each port on an IBA Router shall comply with the physical layer requirements specified in Chapter 6: Physical Layer Interface on page 163.

C19-12: Each port on an IBA Router shall comply with the link layer requirements specified in Chapter 7: Link Layer on page 167 of this specification.

C19-13: Each port on an IBA Router shall implement the Subnet Management PORTINFO Attribute specified in 14.2.5.6 PortInfo on page 821.

C19-14: Each port on an IBA Router shall have at least one GID assigned to it.

19.2.4.3 RECEIVER QUEUING

The receiver queueing function receives packets from the link layer defined in Chapter 7: Link Layer on page 167.

C19-15: The virtual lane into which an individual packet is queued shall be the virtual lane whose virtual lane number matches the VL field in the packet's Local Route Header.

C19-16: If the FilterRawInbound component of the receiving port's PortInfo attribute is set to one, then the Router shall discard all packets received on that port in which the LNH field of the LRH contains a binary 00 or binary 01 (i.e. raw packets).

19.2.4.3.1 INBOUND P_KEY ENFORCEMENT

The implementation of the inbound P_Key enforcement in Routers is optional. This section defines its requirements if implemented.

Inbound P_Key verification shall be enabled and disabled for each port individually based on the PartitionEnforcementInbound component of the PortInfo attribute.

o19-4: If a router provides the inbound P_Key enforcement service and the PartitionEnfocementInbound component of the PortInfo Attribute is set to zero, then the inbound P_key enforcement service shall be disabled for packets received on the corresponding port.

o19-5: If a router provides the inbound P_Key enforcement service, it shall maintain a separate list of P_Keys associated with each port.

o19-6: If a router provides both the inbound P_Key enforcement service and the outbound P_Key enforcement service, then the list of P_Keys associated with each port shall be the same list for both the inbound P_Key enforcement service and the outbound P_Key enforcement service.

o19-7: If a router provides the inbound P_Key enforcement service, the P_Key table associated with each port shall be capable of containing between 1 and 65535 P_Keys, inclusive (the exact number is left as an implementation parameter).

o19-8: If a router provides the inbound P_Key enforcement service, the P_Key table associated with each port shall be programmable using the P_KeyTable attribute defined in 14.2.5.7 P_KeyTable on page 834.

o19-9: If a router provides the inbound P_Key enforcement service and if the PartitionEnfocementInbound component of the PortInfo Attribute is set to one, then any packet received on a virtual lane other than 15 shall either be discarded or truncated such that it contains no data past the BTH

if the value in the P_Key field in the BTH is not contained in the receiving port's P_Key list and either of the following conditions are true:

- LNH field in the LRH contains binary 11 and IPVer field in the GRH contains 6.
- LNH field in the LRH contains binary 10.

o19-10: If a router provides the inbound P_Key enforcement service and if the PartitionEnfocementInbound component of the PortInfo Attribute is set to one, then any packet received on a virtual lane other than 15 shall either be discarded or truncated in length such that it contains no more than 64 bytes if all of the following conditions are true:

- LNH field of the LRH contains binary 11.
- IPVer field of the GRH does not contain 6.

Raw packets are not subject to P_Key enforcement and shall not be discarded nor truncated by this mechanism.

19.2.4.4 PACKET RELAY

Packet relay refers to the operation of transferring a packet from the virtual lane on the inbound port to the virtual lane on a outbound port. The output virtual lane selection is specified later in this section. The relay function is performed regardless of the state of the destination port. In certain port states the destination port will discard the packet.

C19-17: A router shall relay each unicast packet from the virtual lane zero through fourteen (if implemented) in which it was received to the output port indicated by the routing table entry corresponding to the packet's DGID field.

C19-18: Packets received on virtual lane 15 shall not be relayed to output ports.

A packet may be relayed to the same port on which it was received, this is necessary to support some routing scenarios like endnodes using one out of several routers on the subnet as a default router.

o19-11: Routers that support more than one virtual lane, in addition to virtual lane 15, shall set the value of the VL field in the local route header by first considering the GRH Tclass field to derive a SL value for the subnet attached to the output port and then using the SL to VL mapping scheme as defined in section 7.6.6 VL Mapping Within a Subnet on page 186.

C19-19: Routers shall always recognize and map a Tclass value of 0 to a best effort SL.

C19-20: Each packet relayed from an inbound port shall be placed on the virtual lane of the outbound port specified by the SL to VL mapping. If the new value of VL does not correspond to a configured VL on the outbound port, the packet shall be discarded. Also, packets received with a VL not configured for the port shall be discarded.

C19-21: If the virtual lane on the outbound port does not contain sufficient space for the packet to be relayed, then the packet shall remain in the virtual lane on the inbound port until sufficient space is available or until the router lifetime limit mechanism permits the discard of the packet.

C19-22: If the relay function is unable to relay packet from an inbound port to an outbound port due to lack of sufficient space in the outbound VL, the relay function shall continue to relay packets from other virtual lanes destined for virtual lanes on outbound ports with sufficient space.

C19-23: Packets shall be transmitted on a given port and SL in the same order as they were received from a given port.

o19-12: The relay function may, but is not required to, relay packets in the inbound portion of virtual lanes that are behind packets that are blocked due to insufficient space in the outbound portion of virtual lanes.

The method of arbitration when multiple inbound VLs have packets destined for the same outbound VL is left to the implementor, but the arbitration should service all inbound ports fairly.

C19-24: Routers shall not continuously assert backpressure (i.e. fail to grant link credits). Regardless of what congestion policy an IBA router associates to its relay function, routers shall not cause deadlock in the fabric.

C19-25: Packets whose Hop count is less than 2 shall be discarded.

C19-26: The Hop count of every relayed packet is decremented by one.

19.2.4.5 TRANSMITTER QUEUING

Relayed packets shall be queued in the outbound portion of virtual lanes.

19.2.4.5.1 OUTBOUND P_KEY ENFORCEMENT

The implementation of the outbound P_Key enforcement service in routers is optional. This section specifies the requirements of the service if implemented.

Outbound P_Key verification shall be enabled and disabled for each port individually based on the PatrtitionEnforcementOutbound component of the PortInfo attribute.

o19-13: If a router provides the outbound P_Key enforcement service and the PartitionEnfocementOutbound component of the PortInfo Attribute is set to zero, then the outbound P_key enforcement service shall be disabled for packets received on the corresponding port.

o19-14: If a router provides the outbound P_Key enforcement service, it shall maintain a separate list of P_Keys associated with each port.

o19-15: If a router provides both the inbound P_Key enforcement service and the outbound P_Key enforcement service, then the list of P_Keys associate with each port shall be the same list for both the inbound P_Key enforcement service and the outbound P_Key enforcement service.

o19-16: If a router provides the outbound P_Key enforcement service, the P_Key table associated with each port shall be capable of containing between 1 and 65535 P_Keys, inclusive (the exact number is left as an implementation parameter).

o19-17: If a router provides the outbound P_Key enforcement service, the P_Key table associated with each port shall be programmable using the P_KeyTable attribute defined in 14.2.5.7 P_KeyTable on page 834.

o19-18: If a router provides the outbound P_Key enforcement service and if the PartitionEnfocementOutbound component of the PortInfo Attribute is set to one, then any packet to be transmitted on a virtual lane other than 15 on that port shall either be discarded or truncated such that it contains no data past the BTH if the value in the P_Key field in the BTH is not contained in the transmitting port's P_Key list and either of the following conditions are true:

- LNH field in the LRH contains binary 11 and IPVer field in the GRH contains 6.
- LNH field in the LRH contains binary 10.

o19-19: If a router provides the outbound P_Key enforcement service and if the PartitionEnfocementOutbound component of the PortInfo Attribute is set to one, then any packet to be transmitted on a virtual lane other than 15 of that port shall either be discarded or truncated in length such that it contains no more than 64 bytes if all of the following conditions are true:

- LNH field of the LRH contains binary 01 or binary 11.
- IPVer field of the GRH does not contain 6.

Raw packets, i.e. packets in which the LNH field of the LRH contains binary 00 or 01, are not subject to P_Key enforcement and are not discarded nor truncated by this mechanism.

19.2.4.6 PACKET TRANSMISSION

C19-27: If the FilterRawOutbound component of the transmitting port's PortInfo attribute is set to one, then the router shall discard all packets to be transmitted on that port in which the LNH field of the LRH contains a binary 00 or binary 01 (i.e. raw packets).

C19-28: Routers shall perform SL to VL mapping as defined in 7.6.6 VL Mapping Within a Subnet on page 186. This mapping is based on the outbound SL to be used for the packet.

C19-29: Packet shall be transmitted with a valid VCRC field computed as specified in section 7.8.2 Variant CRC (VCRC) - 2 Bytes on page 197, unless required otherwise in this chapter.

C19-30: Each packet shall be transmitted with an EGP character appended unless required otherwise in this chapter.

Routers shall support the requirements specified in 7.6.9 VL Arbitration and Prioritization on page 188.

19.2.5 ERROR HANDLING

This section specifies required operation under error conditions for each of the conceptual functions.

19.2.5.1 ROUTER PORTS ERRORS

C19-31: Each port on an IBA router shall comply with the physical layer error requirements specified in Chapter 6: Physical Layer Interface on page 163.

C19-32: Each port on an IBA router shall comply with the link layer error requirements specified in Chapter 7: Link Layer on page 167 of this specification.

19.2.5.2 RECEIVER QUEUING ERRORS

The receiver queueing function may discard any packet that meets any of the following conditions:

- There is insufficient space in the virtual lane to receive a packet of the size indicated in the PktLen field in the local route header.
- The size of the packet indicated by the PktLen field in the local route header indicates that the packet exceeds the MTU size supported by the Router port.

The receiver queueing function may discard any packet if its transmission has not been initiated and if the packet meets any of the following conditions:

- There is insufficient space in the virtual lane to receive the packet.
- The packet exceeds the MTU size supported by the output port.
- A VCRC error was detected on reception.
- The length of the received packet was different from that indicated by LRH:PktLen.
- The packet has a framing error.
- The packet was received with an EBP delimiter appended.
- The length of the packet was too short to contain a LRH, GRH, and a VCRC.

C19-33: Any packet that exceeds the MTU size supported by the output port and that is not discarded shall be truncated to any size that meets the MTU size limitation of the port.

19.2.5.3 PACKET RELAY ERRORS

C19-34: Packets with no GRH, or with a GRH version not supported by the Router shall be discarded.

19.2.5.4 TRANSMITTER QUEUEING ERRORS

C19-35: Routers shall implement the Packet Lifetime limits and Head of Queue Lifetime Limit mechanisms defined for IBA Switches in 18.2.5.4 Transmitter Queueing on page 1057.

The Packet Lifetime limit is determined from the LifeTimeValue component of the RouterInfo attribute using the same formula as the Switch Lifetime Limit. The Head of Queue Lifetime Limit is determined from the HOQLife component of the PortInfo attribute using the same formula as its switch counterpart.

The above mentioned limits on packet lifetime inside IBA routers and switches are meant to help drain packets from the IBA fabric before they can present a hazard to the IBA transport layer finite sequence number space. These limits are not defined as congestion management mechanisms, and should not be reached in normal circumstances, even in congestion scenarios.

19.2.5.5 PACKET TRANSMISSION ERRORS

C19-36: Each packet to be transmitted that was received with an error indicated by the link layer shall be transmitted with an EBP character appended and the VCRC field shall contain the one's complement of the valid VCRC.

o19-20: Each packet to be transmitted that was received with an error indicated by the link layer may be truncated in length.

C19-37: Each packet to be transmitted that is truncated in length as permitted or specified by any condition in this chapter be corrupted as specified in 7.3 Packet Receiver States on page 172.

19.2.6 SUBNET MANAGEMENT AGENT REQUIREMENTS

C19-38: Each router port shall implement a Subnet Management Interface (SMI) as specified in [Chapter 14: Subnet Management on page 794.

C19-39: Routers shall support a General Services Interface (GSI) as specified in Chapter 16: General Services on page 930.

The General Services Interface is used, for example, for GID to LID address resolution.

CHAPTER 20: VOLUME 1 COMPLIANCE SUMMARY

20.1 COMPLIANCE DEFINITION

This chapter specifies the Compliance Categories that are approved for labeling various products that contain InfiniBand content. This will allow vendors to label their products and claim InfiniBand compliance without creating confusion in the marketplace. This chapter addresses compliance to the feature set defined by Volume 1 of the InfiniBand Specification.

20.1.1 PRODUCT APPLICATION

Each product that has InfiniBand content **may** claim InfiniBand Compliance to one or more of the Categories defined in the Compliance Summaries of the InfiniBand Specification. A product **shall not** simply claim "InfiniBand Compliant".

Each claim of compliance **shall** be a list of one or more valid InfiniBand Compliance Categories from Volume 1 or Volume 2. It's appropriate for some products to include Compliance Categories from both Volumes 1 and 2.

The valid Volume 1 Compliance Categories are defined below.

Those for Volume 2 are defined in *InfiniBand Architecture Specification, Volume 2* Chapter "Volume 2 Compliance Summary".

20.2 VOLUME 1 COMPLIANCE CATEGORIES

Volume 1 Compliance Categories refer to the functionality of each entity defined in Volume 1. Table 316 on page 1073 lists all valid Volume 1 Compliance Categories along with their full names.

Because optional functionality may be associated with a given Compliance *Category*, zero or more Compliance *Qualifiers* may be associated with that Category. Table 316 lists all valid Qualifiers under each Category. Qualifiers shown in ***bold italics*** indicate functionality that is actually non-optional for that specific category, but those Qualifiers may still appear in some of the Compliance Statements listed under that Category.

Table 317 on page 1074 lists Volume 1's complete set of Qualifiers along with their full names. Section 20.2.1 discusses Qualifiers in more depth.

Each Category has a dedicated section in this chapter that contains, among other things, a complete reference list of Volume 1 compliance

statements that directly apply to that category. Table 316 provides a reference to each section, including a hypertext link with on-line versions of the spec.

Table 316 Volume 1 Compliance Categories

Category	Full Name	Valid Qualifiers	Reference
HCA-CI	Host Channel Adapter - Channel Interface	VLs, ***RC***, ***UC***, RD, RawD, APM, UDMcast, RawDMcast, ***RDMA***, Atomics, P_Key traps, P_Key counters, Notice, Trap, BMM, BQM, SRQ, LIF, BL, ZBVA, SDP, VE, OptPC	Section 20.3 on page 1077
TCA	Target Channel Adapter	VLs, RC, UC, RD, RawD, APM, UDMcast, RawDMcast, RDMA, Atomics, P_Key traps, P_Key counters, Notice, Trap, OptPC	Section 20.4 on page 1093
SW	Switch	VLs, UDMcast, P_Key SRE, P_Key SRE_In, P_Key SRE_Out, Notice, Trap, P_Key SEPT, OptPC	Section 20.5 on page 1102
RTR	Router	VLs, RawD, UDMcast, RawDMcast, P_Key SRE, Notice, Trap, RMPP, OptPC	Section 20.6 on page 1106
SM	Subnet Manager	Trap, DRN, SysG, ReIn	Section 20.7 on page 1110
SA	Subnet Administration	UDMcast, Trap, SAOPT, RMPP, MPath	Section 20.8 on page 1112
CM	Communication Manager	APM	Section 20.9 on page 1114
PFM	Performance Manager	Trap, Notice	Section 20.10 on page 1114
VM	Vendor-Defined Manager	Trap, Notice, RMPP	Section 20.11 on page 1115
OMA	Optional Management Agent	Trap, Notice, AMA, DMA, SNMP, VMA	Section 20.12 on page 1116

20.2.1 VOLUME 1 COMPLIANCE QUALIFIERS

Compliance Qualifiers indicate which compliance statements apply only if a product supports an optional feature or specified combination of optional features.

Some compliance statements apply to multiple Compliance Categories, and thus appear in the Compliance Statement List under each applicable Category. Some of these “shared” compliance statements include Qualifiers associated with functionality that is optional in some Categories and mandatory in others. In each Category where the functionality is mandatory, the associated Qualifier is shown in ***bold italics*** for that Category’s “Valid Qualifier’s” entry in Table 316.

20.2.1.1 CLAIMING SUPPORT FOR OPTIONAL FEATURES

C20-1: If a product claims to support a given optional feature, the product **must** comply with **all** compliance statements that apply to that optional feature.

For example, an HCA-CI that claims to support Reliable Datagram Service must comply with all statements under the HCA-CI Compliance Statement List that apply, given the RD Qualifier.

A product **shall not** include in its list of supported *optional* features any features that are in fact *mandatory* for the Category the product claims compliance to. Qualifiers for these mandatory features are shown in ***bold italics*** in Table 316. For example, Reliable Connection Service is mandatory for HCA-CI, so RC must not be included in an HCA-CI's list of supported optional features even though the product must still meet all RC requirements.

A product may claim support for multiple optional features, in which case the product must comply with all compliance statements that apply to the particular set of optional features claimed by the product, noting that some compliance statements apply only for specific combinations of qualifiers.

Table 317 lists and describes the Volume 1 Compliance Qualifiers that a product can claim compliance to. To abbreviate the optional support, one or more Qualifiers can be listed after the Category in braces. For example, a Target Channel Adapter that supports Reliable Datagram Service and also Automatic Path Migration can be abbreviated with TCA{RD,APM}

Table 317 Volume 1 Compliance Qualifiers

Qualifier	Description
AMA	Application-specific Management Agent
APM	Automatic Path Migration
Atomics	Atomic Operations
BL	Block List PBL
BMM	Base Memory Management Extensions
BQM	Base Queue Management Extensions
DM	Device Manager
DMA	Device Management Agent
DRN	Directed Route Notices
LIF	Local Invalidate Fencing
MPath	Multipath requests

Table 317 Volume 1 Compliance Qualifiers (Continued)

Qualifier	Description
Notice	Standard Format & Queue for Data About Events
OptPC	Optional Performance Counters
P_Key counters	Counters for P_Key Violations
P_Key SEPT	P_Key Switch External Port Trap
P_Key SRE	P_Key Enforcement by Switches or Routers
P_Key SRE_In	Inbound P_Key Enforcement by Switches or Routers
P_Key SRE_Out	Outbound P_Key Enforcement by Switches or Routers
P_Key traps	Trap Generation for P_Key Violations
RawD	Raw Datagram Service
RawDMcast	Raw Datagram Multicast
RC	Reliable Connection Service
RD	Reliable Datagram Service
RDMA	Remote Direct Memory Access
ReIn	Reinitialization
RMPP	Reliable Multi-Packet Protocol
SAOPT	Subnet Administration Bulk Update Facilities
SDP	Sockets Direct Protocol
SNMP	SNMP Tunneling Agent
SRQ	Shared Receive Queue
SysG	System GUID
Trap	Asynchronous Event Notification
UC	Unreliable Connection Service
UDMcast	Unreliable Datagram Multicast
VE	Any of the Verb Extensions
VLs	Port Supporting More than One Data VL
VMA	Vendor-specific Management Agent
ZBVA	Zero Based VA

If an optional compliance statement does not contain a valid qualifier, refer to the text of the optional compliance statement to determine its applicability. The text of a compliance statement always takes precedence over the compliance qualifier.

20.2.1.2 COMPLIANCE STATEMENTS WITH MULTIPLE QUALIFIERS

Some compliance statements contain combinations of Qualifiers, and apply only if the specified combination is true. For example, a compliance statement beginning with “RD and Atomics:” applies only if *both* RD and Atomics are supported. If a compliance statement begins with “RD or Atomics:”, the statement **shall** apply if *either* RD or Atomics is supported.

20.2.2 COMPLIANCE STATEMENT LISTS

Within each Compliance Category section is a list of the compliance statements that apply to that particular category. Here is a sample list entry:

- **o9-16:** RD: PSN Insertion for Reliable Svc Pkts. . . . . Page 231

20.2.2.1 HYPERTEXT LINKS

Online versions of this specification have hypertext links present before each of the lines in the Compliance Statement lists. These links are indicated by the “●” at the beginning of the line and will lead to the actual statement in the body of the specification that contains the details for each of the compliance entries.

Each Compliance Statement List entry also contains the page number for use with hard-copy versions of the specification.

20.2.2.2 COMPLIANCE STATEMENT LABELS

All formal compliance statements throughout the specification are labeled so they can be uniquely identified. Each label begins with either a “C” or an “o”, indicating whether the compliance statement applies in all cases with respect to its category or whether the compliance statement is qualified with respect to optional features. The “o” is uncapitalized to make it more easily distinguishable from the “C” in Compliance Statement Lists.

The next portion of the label is the number of the chapter in which the formal compliance statement appears. The final portion of the label is a compliance statement number, which starts with “1” in each chapter. “C” and “o” compliance statements are numbered independently.

20.2.2.3 COMPLIANCE STATEMENT TITLES

Each line within a Compliance Statement List contains a brief title for the respective compliance statement. Because of the limited space and lack of context, each title is only intended to convey the topic of the compliance statement, and not necessarily convey its actual requirements.

Compliance statements that apply only to optional functionality is indicated by the presence of one or more Qualifiers at the beginning of the title, followed by a colon. For example, the above sample Compliance Statement Title contains the “RD” qualifier.

20.2.3 COMMON REQUIREMENTS

Some Compliance Categories share common requirements, such as those that apply to all ports. To avoid unnecessary duplication, certain common requirement sets have been collected and referenced by the appropriate Compliance Categories instead of replicating those lists of requirements under each separate Category.

20.3 HCA-CI COMPLIANCE CATEGORY

In order to claim compliance to the InfiniBand Volume 1 specification to the Compliance Category of HCA-CI, a product **shall** meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support. In addition, a compliant HCA-CI **shall** meet all Section 20.13 Common Port Requirements on page 1117 and all Section 20.14 Common MAD Requirements on page 1119.

Some compliance statements in the HCA-CI Category contain requirements that apply to both mandatory and optional features. For instance, some compliance statements mention both QPs and EE Contexts, though EE Contexts are relevant only if RD Service is supported. In such cases, the requirements on an optional feature apply only if the product claims to support the optional feature.

20.4 TCA COMPLIANCE CATEGORY

In order to claim compliance to the InfiniBand Volume 1 specification to the Compliance Category of TCA, a product **shall** meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support. In addition, a compliant TCA **shall** meet all Section 20.13 Common Port Requirements on page 1117 and all Section 20.14 Common MAD Requirements on page 1119.

20.5 SWITCH COMPLIANCE CATEGORY

In order to claim compliance to the InfiniBand Volume 1 specification to the Compliance Category of Switch, a product **shall** meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support. In addition, a compliant Switch **shall** meet all Section 20.13 Common Port Requirements on page 1117 and all Section 20.14 Common MAD Requirements on page 1119.

20.6 ROUTER COMPLIANCE CATEGORY

In order to claim compliance to the InfiniBand Volume 1 specification to the Compliance Category of Router, a product **shall** meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support. In addition, a compliant Router **shall** meet all Section 20.13 Common Port Requirements on page 1117 and all Section 20.14 Common MAD Requirements on page 1119.

20.7 SUBNET MANAGER COMPLIANCE CATEGORY

In order to claim compliance to the InfiniBand Volume 1 specification to the Compliance Category of Subnet Manager, a product **shall** meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support. In addition, a compliant Subnet Manager **shall** meet all Section 20.14 Common MAD Requirements on page 1119.

20.8 SUBNET ADMINISTRATION COMPLIANCE CATEGORY

In order to claim compliance to the InfiniBand Volume 1 specification to the Compliance Category of Subnet Administration, a product **shall** meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support. In addition, compliant Subnet Administration **shall** meet all Section 20.14 Common MAD Requirements on page 1119.

20.9 COMMUNICATION MANAGER COMPLIANCE CATEGORY

In order to claim compliance to the InfiniBand Volume 1 specification to the Compliance Category of Communication Manager, a product **shall** meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support. In addition, a compliant Communication Manager **shall** meet all Section 20.14 Common MAD Requirements on page 1119.

Though a number of optional CM-specific features exist, no CM-unique qualifiers have been defined since the optional CM-specific features are fully described within the CM Chapter, and most of the chapter's optional compliance statements would require unique qualifiers.

20.10 PERFORMANCE MANAGER COMPLIANCE CATEGORY

In order to claim compliance to the InfiniBand Volume 1 specification to the Compliance Category of Performance Manager, a product **shall** meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support. In addition, a compliant Performance Manager **shall** meet all Section 20.14 Common MAD Requirements on page 1119.

20.11 VENDOR-DEFINED MANAGER COMPLIANCE CATEGORY

In order to claim compliance to the InfiniBand Volume 1 specification to the Compliance Category of Vendor-Defined Manager, a product **shall** meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support. In addition, a compliant Vendor-Defined Manager **shall** meet all Section 20.14 Common MAD Requirements on page 1119.

20.12 OPTIONAL MANAGEMENT AGENT COMPLIANCE CATEGORY

In order to claim compliance to the InfiniBand Volume 1 specification to the Compliance Category of Optional Management Agent, a product **shall** meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support. In addition, a compliant Optional Management Agent **shall** meet all Section 20.14 Common MAD Requirements on page 1119.

20.13 COMMON PORT REQUIREMENTS

Multiple Compliance Categories share common Port Requirements. To avoid unnecessary duplication, Port Requirements are collected here and referenced by the appropriate Compliance Categories.

20.14 COMMON MAD REQUIREMENTS

Multiple Compliance Categories share common Management Datagram Requirements. To avoid unnecessary duplication, Management Datagram Requirements are collected here and referenced by the appropriate Compliance Categories.

ANNEX A1: I/O INFRASTRUCTURE

A1.1 INTRODUCTION

A1.1.1 PURPOSE

This document is a supplement to Volume 1 of the InfiniBand Architecture, herein referred to as the base document.

It is an informative annex and does not contain any compliance requirements. This annex describes the InfiniBand I/O framework and recommends various policies for discovering, managing, and using I/o devices attached via the IB fabric.

A1.1.2 GLOSSARY

This annex uses the following terms in addition to the terms defined in the base document Glossary (Chapter 2).

Compatibility String — An ASCII string used to match an I/O controller with a compatible I/O Device Driver.

I/O Device Driver — Code, typically in a host, that is used to control an I/O device. I/O device drivers allow programs in the host to control I/O devices in a device independent way. An I/O device driver may be a generic driver designed for a standard I/O Protocol regardless of the device vendor, or it may be a vendor-specific driver, for a particular vendor's device or set of devices.

I/O Initiator — An I/O client that sends I/O commands to an I/O device.

I/O Protocol — The set of rules governing the content and exchange of I/O related information passed between host platforms and I/O controllers.

SCSI RDMA Protocol — A Storage Protocol defined by T10 that maps SCSI onto transport protocols (such as IBA) that support remote DMA. It specifically provides an annex that maps SCSI onto IBA.

Service Connection — All the channels that a single initiator establishes with a particular I/O controller.

SRP — SCSI RDMA Protocol

Storage Protocol — An I/O Protocol specifically for storage devices.

A1.2 PRINCIPLES OF I/O

A1.2.1 I/O OPERATION OVERVIEW

The IBA fabric provides a scalable interconnect that supports numerous I/O devices. As an I/O network, IBA supports many different configurations such as direct connected I/O (an I/O unit directly connected to a host's HCA port without a switch), single system (a single host, a switch, and a number of I/O units) as in Figure 227, multiple systems (multiple hosts with I/O units dedicated to each one) as in Figure 228 and other configurations (where I/O devices may be shared among multiple hosts).

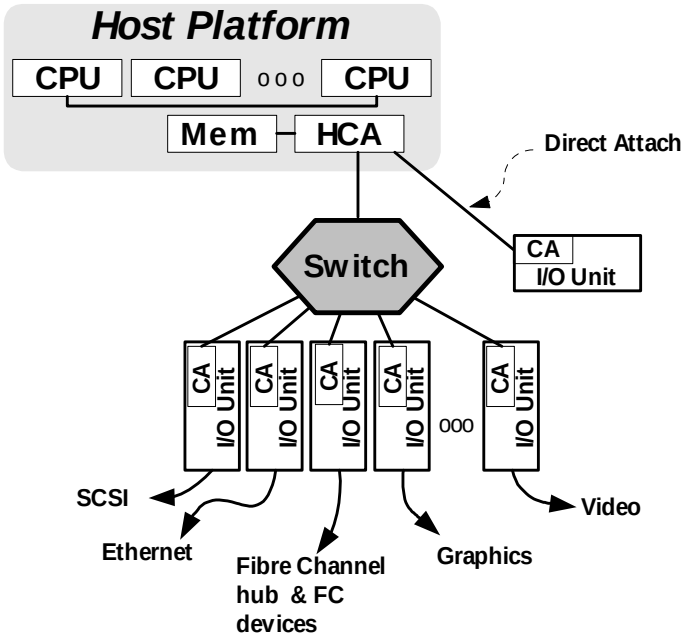


Figure 227 Single System & Direct Attached I/O

I/O units might contain one or more constructs referred to as I/O controllers. Each I/O controller may choose to support only a single initiator (e.g., a host) or might support multiple initiators (either sequentially or concurrently). Each I/O controller may be simple (i.e., provide a single QP for each initiator) or complex and use multiple channels with each initiator. The term service connection refers to all the channels that a single initiator establishes with an I/O controller. When an I/O controller cannot support an additional initiator, the IOU's CM rejects CM:REQ messages with an appropriate reject code (e.g., No QP Available, No Resources Available, Invalid ServiceID, and Consumer Reject).

An I/O unit is composed of a channel adapter (HCA or TCA) and a number of I/O controllers. A platform that is a host might also provide I/O capabilities in which case that host is also considered an I/O unit. A managed I/O

unit is an I/O unit that supports Device Management; the mechanism an I/O unit uses to deliver information about the I/O unit's I/O resources. Class version 1 of device management is specified in the base document section 16.3 "Device Management". Class version 2 is specified in Annex A8: Device Management.

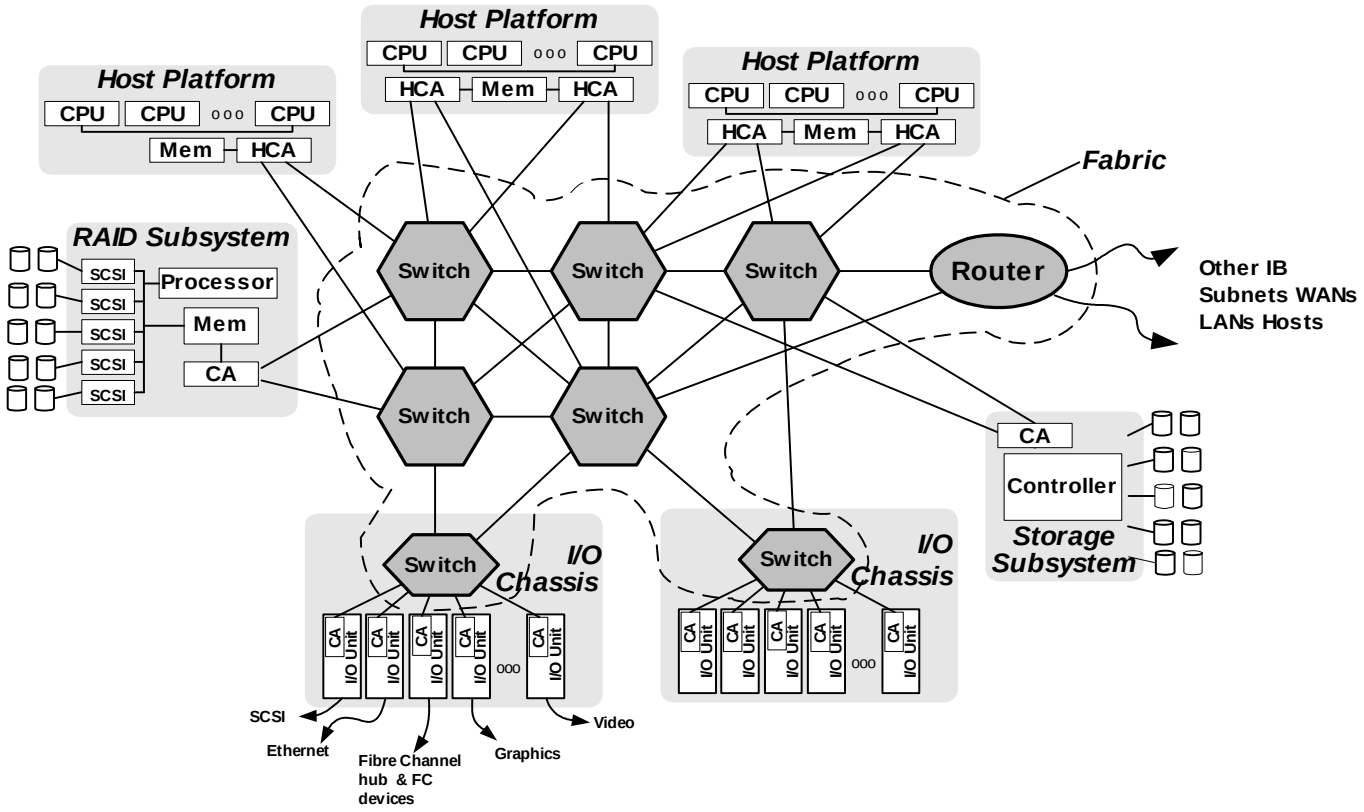


Figure 228 Multiple Systems

I/O operation is independent of subnet complexity. The host enumerates I/O controllers within a managed I/O unit by querying the I/O unit. A host may discover I/O units in its partitions by querying Subnet Administration (SA) and/or the Configuration Manager (CFM). A CFM, if it exists, provides information about I/O controllers to particular hosts. The host may query the CFM to retrieve a list of its I/O controllers. The host can register with the CFM to be notified about new arrivals and removals. The CFM is defined in the "Configuration Management" annex. Specific use of I/O controllers for booting is captured in the Booting Annex and is outside the scope of this annex.

The host may retrieve information about each I/O controller in a managed I/O unit using DevMgtGet MADs. Using controller and protocol information from the DevMgtGet()s, the host is able to match an I/O device driver to

that controller. The host then invokes the I/O device driver which establishes necessary connections with the I/O controller and enumerates I/O ports and/or devices behind the I/O controller. At this point, operation is strictly in the hands of the I/O device driver and outside the scope of the IB specification.

A1.2.2 MANAGED I/O UNITS

IB Device Management class specifies methods and attributes used for managing IB devices that provide I/O. IB devices managed by these device management methods and attributes are referred to as managed I/O units.

The architectural model of a managed I/O unit is illustrated in Figure 229.

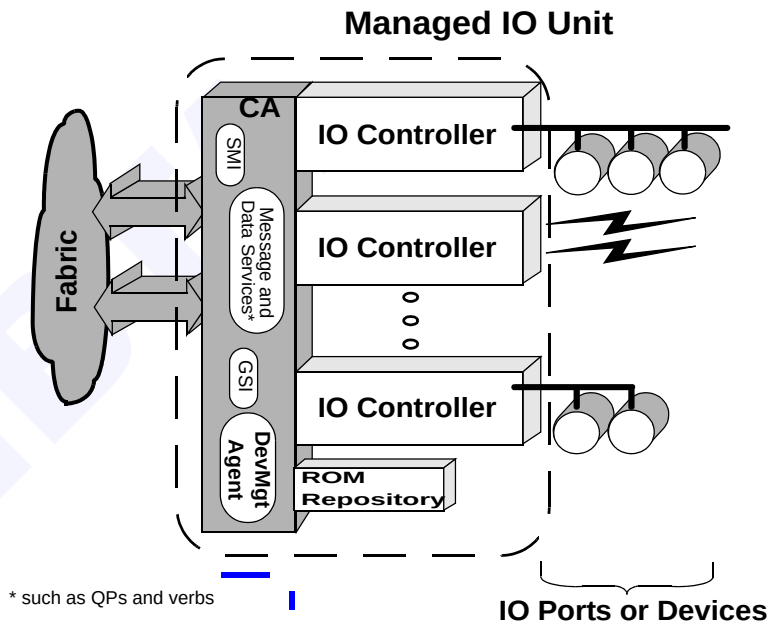


Figure 229 Architectural Model for a Managed I/O Unit

The Device Management Class specifies messages that can be used for determining the number of I/O controllers for a managed I/O unit and the existence of an optional ROM repository. Using these methods and attributes, a host can determine the protocols and services supported by each I/O controller, as well as other capabilities of the I/O controllers. See chapter 16 and Annex A8: for additional information.

A1.2.3 ROM REPOSITORY

The ROM repository, illustrated in Figure 229, provides a means for managed I/O units to supply I/O device drivers for controlling IOCs during booting. The ROM repository may also contain other information such as

downloaded microcode or data related to initializing and using IOCs. For additional information, see Boot Management Annex.

A1.2.4 I/O DEVICE DRIVERS

I/O Device Drivers are code images which drive I/O controllers. An I/O device driver may be vendor-specific for a particular vendor's I/O controller or set of similar I/O controllers, or it may be a standard I/O device driver designed for a standard I/O protocol regardless of the I/O controller vendor.

This section describes the recommended practice for a device manager to match an I/O controller with an appropriate I/O device driver. This is the practice mandated for matching boot drivers in the ROM repository (see Booting Annex).

Each I/O device driver image, regardless of whether or not it is stored in a ROM repository, is associated with a set of character strings which identify the I/O protocols and/or vendor products which the I/O device driver supports. These character strings are called "compatibility strings." An I/O device driver image may be associated with one or more compatibility strings, depending on the number of protocols and/or vendor products which it supports. These compatibility strings allow the I/O device driver to be matched with I/O controllers with which it can be used.

A1.2.4.1 MATCHING AN I/O CONTROLLER WITH AN I/O DEVICE DRIVER

The process of matching an I/O device driver with an I/O controller involves creating a set of compatibility strings for the I/O controller, and comparing the compatibility strings generated for the I/O controller to the compatibility strings associated with each I/O device driver. If one of the compatibility strings associated with the I/O device driver matches one of the compatibility strings of the I/O controller, then the driver may be used with the I/O controller.

A1.2.4.1.1 CREATING COMPATIBILITY STRINGS FOR AN I/O CONTROLLER

In order to generate the compatibility strings for an IOC, a host sends DevMgtGet()s to the managed I/O unit containing the IOC requesting IOControllerProfile and other pertinent attributes. The attributes contain a set of components identifying the IOC and its vendor (VendorID, IocDeviceID, Device Version, Subsystem VendorID, SubsystemID), and a set of components identifying the protocols supported by the IOC (I/O Class, I/O Subclass, Protocol, and Protocol Version). These components are in binary format. See A3.3 "I/O Controller Identification" on page 1189 for additional details regarding the component values corresponding to various protocols.

The host converts the binary value of each component into its ASCII representation with a leading designator. For example, if the IOControllerPro-

file Device Version component was 0xabcd, which is a 16-bit quantity, then the ASCII string into which it is converted is 'vabcd', which is a 40-bit quantity. Table 318 specifies how the DevMgt attribute components are converted into an ASCII string.

Table 318 ASCII String Representations of DevMgt Components

Component Name	Component Length	ASCII String Representation of Component[a]	ASCII String Length
V - VendorID:	24 bits	Vxxxxxx	56 bits
P - IocDeviceID (product):	32 bits	Pxxxxxxxx	72 bits
v - Device Version	16 bits	vxxxx	40 bits
S - Subsystem VendorID:	24 bits	Sxxxxxx	56 bits
s - Subsystem ID:	32 bits	sxxxxxxxx	72 bits
C - Class:	16 bits	Cxxxx	40 bits
c - SubClass:	16 bits	cxxxx	40 bits
p - Protocol:	16 bits	pxxxx	40 bits
r - Protocol version:	16 bits	rxxxx	40 bits

a. x = hexadecimal character in lower case ASCII (0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f)

The host generates the following six compatibility strings by concatenating the ASCII strings from Table 318 into the compatibility strings of Table 319.

Table 319 Compatibility Strings

		Compatibility String	String Length (Bytes)
Vendor/Product Strings	1	VxxxxxxPxxxxxxxxSxxxxxxsxxxxxxxxvxxxx	37
	2	VxxxxxxPxxxxxxxxSxxxxxxsxxxxxxxx	32
	3	VxxxxxxPxxxxxxxxvxxxx	21
	4	VxxxxxxPxxxxxxxx	16
Class Strings	5	Cxxxxcxxxxpxxxxrxxxx	20
	6	Cxxxxcxxxxpxxxx	15

A1.2.4.1.2 COMPARING COMPATIBILITY STRINGS

The compatibility strings shown in Table 319 serve as product or protocol designators. Since I/O device drivers are written for particular products, family of products, or specific protocols, each compatibility string represents the potential name of a compatible I/O device driver. Thus, they are convenient strings used to unambiguously reference a driver. Each driver can thus be represented by one or more I/O device driver names of the format shown in Table 319 (i.e., one name per product, product family, or protocol that the driver can control). The driver writer selects the I/O device driver name (or names) to express how closely an implementation of an I/O device driver matches a particular product or protocol.

In order to match an I/O device driver with an IOC, the host generates the first string in Table 319 and compares it against the list of I/O device driver names for each I/O device driver. If it finds an exact match between the generated string describing the IOC and an entry in the list of I/O device driver names, then the host selects that driver. If no match is found, the host repeats the process by generating and comparing the second string (and then each of the following strings) until a it finds a match. If no match is found between any of the generated strings and any I/O device driver names, there is no matching driver.

Note that strings in Table 319 on page 1126 are listed from most specific to least specific. The order assumes that an IOC which works with a generic “class” driver (which matches the 5th or 6th compatibility string), will work better with a product-and-version-specific driver matching the first string, if it is present.

The order of search is specifically called out so that the driver matching algorithm will produce deterministic results and select more specific drivers over more generic drivers. This allows I/O device driver writers and hardware vendors to know which drivers will supersede which other drivers in the various different host environments.

A1.2.4.2 USING AN I/O CONTROLLER

Each IOU contains one or more IOCs and each IOC contains one or more I/O objects.The following is a walk-through of the process that a host might perform in order to use an IO object.

To communicate with IOC, the host needs to determine the address (LID and/or GID) of the I/O unit containing the IOC and it needs to know the I/O controller GUID. This information can be obtained in a number of ways. In a fully managed subnet, the host can query the CFM to learn of its I/O objects, otherwise the host needs to remember its I/O objects or “walk the bus” and discover its I/O objects (see A1.3.1 “I/O Device Resolution” on page 1130 for additional information regarding the determination of I/O objects and I/O unit address.)

If the host knows the I/O controller GUID and needs to locate the IOU, the steps differ depending on whether the IOU's Device Management agent supports Class Version 1 or 2.

- For class version 1, the host sends a DevMgtGet(IOUnitInfo) to each IOU in order to obtain a list of I/O slots in the I/O unit. For each slot containing an IOC, the host sends a DevMgtGet(IOControllerProfile) to determine whether the IOC in the requested slot has the proper I/O Controller GUID.

  When a slot containing the IOC with the desired GUID is found, the host uses information in the IOControllerProfile attribute to match the IOC with an I/O device driver. (See A1.2.4.1 "Matching an I/O Controller with an I/O Device Driver" on page 1125.)

  At this point the host has found the IOC and matched it with an appropriate I/O device driver. The I/O device driver needs to establish I/O service with the IOC. This entails the host setting up one or more connections with the IOC for exchanging control and data. However, an IOC might support multiple I/O protocols. The process is to query the IOU via a DevMgtGet(ServiceEneties) and match the service name of the particular I/O protocol to get the ServiceID (used by the CM to connect to a QP). The IOC lists protocols in its service entries list by their Service Name. Thus, the ServiceEntries attribute translates the Service Name to a service ID.

  Each I/O protocol has its own service name and the I/O device driver uses that service name to derive the service ID, which it subsequently uses in a CM:REQ message. At the target, the Service ID distinguishes both the IOC and the protocol.

  Thus, after the appropriate I/O device driver is found, the host sends a DevMgtGet(ServiceEntries) request to the IOC in that slot to obtain a list of ServiceNames and their corresponding Service IDs. The I/O device driver then establishes one or more connections with the IOC using the ServiceID(s) corresponding to the ServiceName(s) specified by the I/O device driver. (Note that the ServiceNames for an I/O device driver is designed into the I/O device driver.)

- For class version 2, the host simply sends a DevMgtGet(ServiceRecord) query to each IOU specifying the particular IOC and service object. And then sends DevMgtGet(IOControllerProfile) and DevMgtGet(ProtorcolRecord) queries to the IOU that responds with the ServiceRecord to get information to match the IOC with an I/O device driver. (See A1.2.4.1 "Matching an I/O Controller with an I/O Device Driver" on page 1125). The ServiceID in the ServiceRecord provides the driver with the information it needs to establish connections with the service object.

For additional details on the device management methods mentioned above, see chapter 16, section 3, Device Management, Annex A7: Configuration Management, and Annex A8: Device Management. For additional details on establishing connections, see chapter 12.

A1.2.5 I/O ATTACHMENT

A1.2.5.1 DIRECT ATTACHMENT

Direct I/O attachment is the attachment of a host IB port to an I/O unit without a switch. In this case, the host provides subnet management (SM). Because the subnet is extremely simple (2 CAs and no switch), the role of the SM is significantly reduced, and optional management capabilities are not expected to be provided.

For a node to support Direct Attached I/O it needs to supply a subnet manager.

A1.2.5.2 FABRIC ATTACHMENT

Fabric complexities range from configurations consisting of a single host, single switch, and multiple I/O units, to configurations consisting of multiple hosts, multiple switches, and multiple I/O units. The I/O units may be either shared by multiple hosts or dedicated to a single host.

As subnets increase in size, it becomes increasingly important for IB devices to support baseboard management functions such as power management. It also becomes more useful for the subnet to provide a Configuration Manager (CFM), Boot Information Service (BIS), and Boot Manager. See related annexes.

A1.2.5.3 POWER MANAGEMENT

Volume 2 and Baseboard Management class provides for power management of nodes attached to the IB fabric including the ability to power down and wake up nodes. I/O operation described herein assumes that power management is transparent to I/O operation and assumes that the I/O units are alive and active.

However, the impact of power management might be noticeable. This is especially true when an I/O unit has been put into a power off state. Assuming that the power manager automatically detects a host's attempt to establish communications with a powered down I/O unit, and immediately powers the I/O unit on, there might be significant delay before the I/O unit is capable of responding to DevMgt and CM MADs.

This delay is not IBA related, but rather a function of platform implementation. Both hosts and I/O units should consider this delay in their designs.

The host should take into consideration that there may be a delay from the time that it starts to send MADs to an I/O unit until the I/O unit is capable

of responding. This delay could include the time for the power manager to respond and the time for the I/O unit to initialize. Currently this time is not bounded, but is expected to be in the order of seconds. For example, the Boot Management class (see Booting Annex) provides a time-out value that the boot manager configures so a booting platform knows the minimum amount of time it needs to wait from the first time it attempts to access an I/O unit before it can claim the I/O unit does not exist.

It is recommended that I/O units that take significant time to initialize, especially those supporting power management and wake-on-fabric, be able to respond with DevMgtGetResp(PortClassInfo) as soon as possible.

A1.3 I/O MANAGEMENT

This section provides an overview on how I/O devices are located and accessed.

A1.3.1 I/O DEVICE RESOLUTION

I/O device resolution is the process of determining the set of nodes which contain I/O devices. There are several resolution methods:

- **Persistent Information:** The host persistently stores information regarding the node containing the I/O device. See Section A1.3.1.2 for additional information.
- **Configuration Manager (CFM):** The host sends a query to the CFM to get a list of I/O objects assigned to the host. See Annex A7: Configuration Management.
- **Subnet Administrator (SA):** The host sends a SubnAdminGetTable(PortInfo) query to the SA in order to obtain the PortInfo attributes of all accessible ports. (See base document chapter 15.) The host filters the list of PortInfo attributes received from the SA to select the ports which have the PortInfo:CapabilityMask:IsDeviceManagementSupported bit set to one. Each port supporting Device Management is then queried to determine if it contains one or more IOCs.
- **Boot Manager**: A boot manager programs a booting node with the identity of the IOU/IOC/ and I/O object that the host uses to boot. See Annex A5: Booting Annex.
- **Boot Information Server (BIS)**: A booting node queries the BIS for the identity of the IOU, IOC, and I/O object that the host uses to boot. See Annex A6: Boot Information Service.

A1.3.1.1 RESOLVING A PATH

During the resolution process, the host determines either the Node GUID, Port GUID, or port GID of the target I/O unit. In order to communicate with that node, however, the host requires a DLID, possibly a DGID, and other

information about the path to the node. The steps in obtaining this information is listed below.

1) If the host knows the Node GUID, it can issue a **SubnAdmGetTable**(**NodeRecord**) query to the SA specifying the NodeGUID of the IOU. This request produces a list of NodeInfo attributes, one for each port on the IOU. (See chapter 15 for additional information.)
2) Select one of the NodeInfo attributes obtained in step 1, and construct a GID by pre-pending the local subnet prefix to the Port GUID component from the NodeInfo attribute.
3) Once the host obtains a Port GID, it can send a **SubnAdmGetTable**(**PathRecord**) query to the SA with the DGID component set to the Port GUID obtained above. Typically, the PathRecord request contains an SGID of the host platform, the DGID equal to the port-GID, and wildcards the SLID to account for a non-zero LMC. The SM returns all paths from the host's port to the I/O unit's port and provides the host with the information (DLID, SL, MTU, etc.) necessary to send packets to the I/O unit.

A1.3.1.2 PERSISTENT INFORMATION

After a node has selected an IOC, it may need to save information about the IOC persistently so that the IOC may be accessed in the future. When this is necessary, the minimum set of information about the IOC which must be persistently stored includes the following:

- **Node GUID** - the Node GUID of the IOU containing the IOC.
- **IOC GUID** - the GUID of the IOC.
- **Additional Information** - protocol-specific information about I/O devices.

See A1.3.1.1 "Resolving A Path" on page 1130 and A1.2.4.2 "Using an I/O Controller" on page 1127 for additional information regarding how the above persistent information is used to access the I/O device.

A1.3.1.3 CONFIGURATION CHANGES

Depending on the upper level protocol, and the way it identifies its target, maintenance scenarios involving the replacement of a failed device may cause the protocol to loose addressability of its target. For instance, if the host's protocol uses the persistent identifiers defined in Section A1.3.1.2 to identify its target, and either the I/O unit or I/O controller must be replaced due to failure, it will be impossible for the protocol to obtain a path using their GUIDs because the new unit (or I/O controller) will have a different GUID. Similarly, if a pair of I/O Units or I/O Controllers are swapped, the protocol might mistakenly swap the service objects behind those IOUs or IOCs. This may or may not be the intended result.

For most upper level protocols, such as SRP, this should not be a problem, since they use other methods of identifying their targets.

In general, if the upper level protocol uses persistent identifiers to locate its target, the recommended solution is simply to change the identifier in the protocol configuration definition file. This will eliminate any ambiguities and assure that the configuration definition file is consistent with the fabric.

If a protocol using persistent identifiers requires the ability to gracefully handle replacement scenarios, the following subsections provide some guidance. None of these strategies are mandatory. None of these strategies are mandatory. In addition, if the protocol does not update its configuration definition file with the new identifier upon detecting such a change, the protocol will be forced to go through this rediscovery processing each time the system is re-booted.

The following subsections also provide insight for how a configuration manager can deal with an I/O unit or I/O controller being replaced or swapped.

A1.3.1.3.1 IDENTIFIERS SUPPORTING CONFIGURATION CHANGES

In order to gracefully handle replacement scenarios, storage of the following identifiers is recommended:

- **ChassisGUID:** global ID of the chassis containing the node

  (See Volume 2, section 13.7.7, ChassisInfo Record:ChassisGUID.)
- **Module GUID:** global ID of module containing the node

  (See Volume 2, section 13.7.6, ModuleInfo Record:ModuleGUID.)
- **SlotNumber:** the IB Module slot to which the IB module containing the node is attached

  (See Volume 2, section 13.7.7, ChassisInfo Record:SlotNumber.)
- **NodeString:** locally administered node name

  (See Chapter 14, section 14.2.5.2, NodeDescription.)
- **I/O Controller Slot Number:** position of IOC in the ControllerList component of the IOUnitInfo attribute of the Device Management agent. (See Chapter 16, section 16.3.3.3, IOUnitInfo and A8.3.3.5: IOUnitInfo.)

A1.3.1.3.2 NODE REPLACEMENT

When the host can not locate an I/O unit with a desired Node GUID, it might mean that a channel adapter, module, and/or chassis has been replaced. When this occurs, the host might be able to determine the Node GUID of a possible replacement. Node replacement strategy is a an im-

plementation policy set by each OS vendor. Some of the possibilities might include:

- Make no attempt to associate a new I/O unit with a missing one.
- Select a node with the same ModuleGUID. Such a node can be found by searching the subnet for any node with the same ModuleGUID. This covers the case in which the desired channel adapter has been replaced within the same module. Note that if the module contains multiple channel adapters, there is a risk that an incorrect channel adapter will be selected.
- Select a node with the same NodeDescription:NodeString component. This can be done by querying the subnet administrator for a list of nodes with the same NodeDescription:Nodestring component as the missing I/O unit. Note that if the subnet was configured with multiple I/O units containing the same NodeString, then there is a risk that an incorrect I/O unit will be selected.
- Select a node with the same ChassisGUID and SlotNumber. Such a node can be found by searching the subnet for any node (with the same ChassisGUID and SlotNumber). This covers the case in which the module containing the I/O unit was replaced. Note that if the module contains multiple I/O units, there is a risk that an incorrect I/O unit will be selected. There is also a risk that the desired module was replaced with a totally unrelated module.

In addition to the above possibilities, it may be appropriate to search for an IOU containing an IOC with the same IOCGUID. This covers the case in which the desired IOC has been moved.

In any of the above cases there is a risk that either the desired I/O device will not be found, or an inappropriate I/O device will be found. Therefore, operator confirmation is recommended before proceeding with the device access.

A1.3.1.3.3 IOC REPLACEMENT

The recommendation to I/O unit vendors producing I/O units with removable I/O controllers is that the I/O controller slot number specified in the IOUnitInfo and IOControllerProfile attributes be associated with its physical location. This allows an operator to replace an IOC and have the new IOC be located at the same I/O slot number as the old IOC. If the old IOC GUID is not found, then the IOC user (e.g., the OS device manager) can look for the new IOC at the same I/O slot. Note that DevMgt class version 2 supports multiple IOCs per I/O module. Thus, there my be more than one IOC with the same I/O slot number, and therefore there is a risk that an incorrect I/O controller could be selected.

A1.3.2 RETRY-BACKOFF POLICY

The I/O resolution process involves accessing a number of different services using unreliable datagrams, and there are a number of reasons why a request might be lost including random bit errors, congestion, and device or service not ready. In the case of random bit errors, retrying immediately has a high probability of success, however in most situations involving congestion and busy devices, immediate retries are not helpful and may create or add to congestion. Therefore, a host needs a simple algorithm for increasing back-off intervals each time the host encounters a situation where it needs to retry an operation.

When an operation is to be retried, the sender first waits the expected maximum response time for the response. (See Chapter 13, section 13.4.6.3.) If no response is received within the expected maximum response time, the sender retries the operation immediately.

If no response is received for this second attempt, the next retry is performed at a randomly selected back-off time between 1 and 100 msec.

If this retry also fails and if subsequent retries are to be performed, the sender doubles the back-off time used for the previous retry before retrying again.

If the back-off time exceeds 30 seconds, the sender randomly selects another back-off time between 1 and 100 msec and repeats the process.

A1.4 IMPACT OF PARTITIONS ON I/O

When there is only a single partition (e.g., the default partition), life is simple and the host does not have to make any decisions regarding partitions. This section discusses various concerns centering around the use of multiple partitions and their impact on I/O. See Chapter 10 section 10.9 “Partitioning” for a description of partitioning. Partitions are enforced on a per port basis.

The Subnet Manager assigns every port to at least one partition and possibly many partitions. The purpose of each partition varies. Any partition that allows a host to communicate with an I/O Unit is considered an I/O partition. Furthermore, I/O related services such as CFM, BIS, and Boot Manager are also bound by partitioning rules. The Subnet Manager does not provide the host with information concerning the purpose of its partitions. Thus, a host does not know which partitions are I/O partitions.

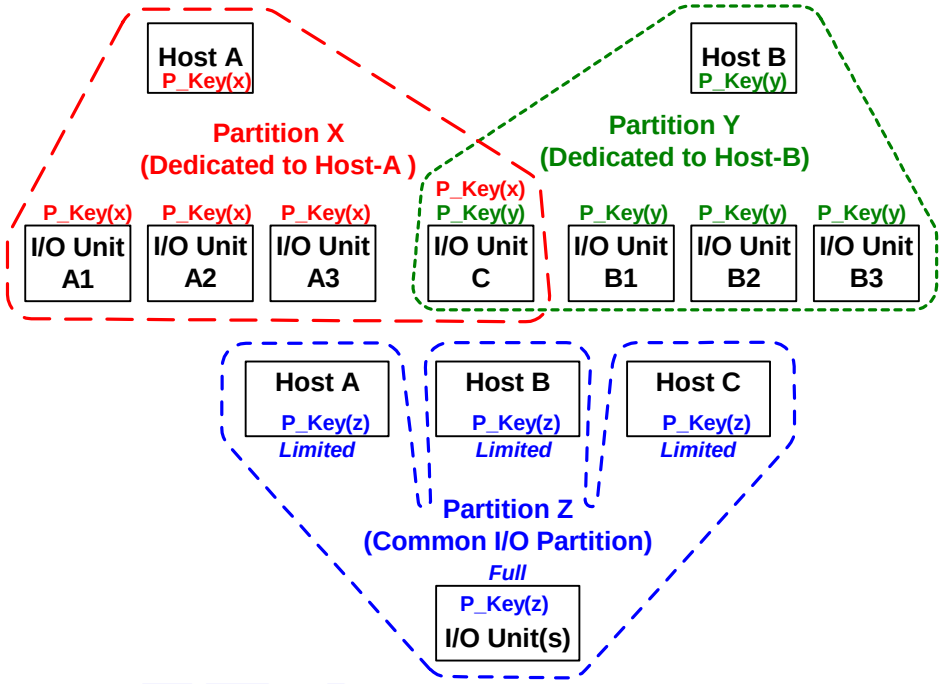


Figure 230 I/O Partitions

An administrator sets up I/O Partitions to enable and enforce access between specified ports (e.g., a host and its I/O units). The sharing pattern of each partition might be a dedicated partition or a common partition as illustrated in Figure 230. Typically the administrator provides at least one I/O partition per host dedicated to that host and assigns I/O units to that partition as illustrated by Partition X and Partition Y.

However, I/O units that need to be accessed by multiple hosts, such as I/O Unit C in Figure 230, need to be a member of each I/O partition that it serves. This means, that an I/O unit accessed by many hosts needs a large P_Key Table. An alternative to requiring I/O units with large P_Key Tables is for the administrator to also provide a common I/O partition for each group of I/O units shared by the same set of hosts, as illustrated by partition Z. In order to maintain separation between hosts in the common I/O partition, the subnet manager configures each host with limited partition membership for that partition while the I/O Units have full membership (see Chapter 10 section 10.9 “Partitioning” subsection titled “Limited and Full Membership”). The point is that hosts might have multiple I/O partitions, one for each relative group of I/O units.

A host containing an I/O related service such as a CFM, BIS, or Boot Manager has to have a partition in common with each host it serves. However,

it is inappropriate to add the host containing the I/O service to an I/O partition if that host is not allowed access to the I/O units. For this case the service would use a non-I/O partition, such as a management partition. The default partition[26] is suitable for the case where these services coexist with the Subnet Manager. Otherwise, the service and the hosts it serves might be assigned to a common management partition (similar to the common I/O partition described above).

The conclusion is that both hosts and I/O units can be assigned to multiple I/O partitions, as well as other partitions. However, the impact of partitioning is not as severe as one might expect. The following sections describe various ways that host and I/O units can deal with multiple partitions.

A1.4.1 I/O UNITS AND PARTITIONS

I/O units are typically targets, that is, hosts make connections to I/O units. The CM:REQ attribute contains a “Partition Key” field that specifies the P_Key for the connection. Thus, an I/O unit simply validates CommMgt MADs based on whether the specified partition is in the port’s P_Key table or not. Only when an I/O unit is to be the initiator of a connection (in this case it is behaving as a host), will it have the same issues as a host. In other words, if the IOU is behaving as a host, the guidance given to hosts in the rest of this section applies to the I/O unit.

A1.4.2 HOSTS AND I/O PARTITIONS

If a host uses the management facilities appropriately, partitions can be semi-transparent to hosts, meaning that the host does not need to treat each partition separately. That is, a host does not have to scan for I/O Units on a partition-by-partition basis -- because discovery mechanisms are outside of partitions. However, I/O access mechanisms are governed by partitioning. Thus, when communicating with I/O units and I/O services, the host does have to use the proper partition, which might be a different partition for each one.

It is the Subnet Administration (SubnAdm) query methods that permits the host to discover I/O devices across all partitions. For example, the host can send a single SubnAdm query to the Subnet Administration agent (SA) such that the SA returns responses for all I/O units that are in a partition for which the host is a member.

A host does need to treat each port independently of the other. If multiple ports attach to the same subnet, then each port is potentially a different path to the same I/O resources. If the ports are on different subnets, then the query subsystem returns information pertinent to that subnet.

26. The default partition purpose, as defined in Chapter 14 and 15 (sections section 14.4 and 15.4), is to permit all nodes to communicate with the SA.

In the process of managing its I/O, there are a number times that a host queries the SA for information. Typically, this involves two SubnAdm methods and several SubnAdm attributes. The methods are SubnAdmGet and SubnAdmGetTable. The attributes are ServiceRecord, NodeRecord, PortInfoRecord, and PathRecord.

The solution is to query the SA and use the Component Mask to indicate that the partition is a wildcard. Since the SA only returns information associated with partitions that are valid for the host (i.e only information about nodes with which the host can communicate), the result is that the SA responds with information for all of the host's partitions and each record returned indicates the partition that the host needs to use.

See Chapter 15 section titled "Administration of Query Subsystem" for details on "querying by ComponentMask". The following are examples of how the host can apply this concept.

A1.4.2.1 QUERY FOR PATH

The host can generate a single SubnAdmGetTable(PathRecord) query and get a list of paths to all other ports that are members of the host's partitions, except for paths where both ports have limited partition membership. This is documented in Chapter 15 section titled "PathRecord" and referred to as "bus walk" because it provides a list of all other nodes the host can access (I/O units and other hosts). From this list, the host can select the paths to its I/O units and the nodes supplying other services such as CFM and BIS.

Each record returned in the response identifies one path to a target port and provides a P_Key that the host can use to communicate with that port. Thus, the partition structure (i.e., whether the host or the I/O Unit is the full member and whether or not there are multiple hosts) is irrelevant to the host.

A1.4.2.2 QUERY FOR SERVICE

To get the list of all nodes providing a particular service, such as BIS or CFM, the host can query the SA with a SubnAdmGetTable(ServiceRecord) specifying a ComponentMask of 0x0000-0000-0000-0020 (bit 6 of ComponentMask = ServiceName component) and specifying the desired service name in the ServiceRecord Attribute. For example, if the host asks for the "BIS.IBTA" service name, then it receives a list of all registered Boot Information Servers that it can reach through that port.

A1.4.2.3 QUERY FOR LIST OF I/O UNITS

A host can derive a list of all managed I/O units on all of its I/O partitions by sending SubnAdmGetTable(PortInfoRecord) query for all records. The SA only returns records for which ports the host can access. The host then

filters those records for ports with the IsDeviceManagementSupported bit of the CapabilityMask component set.

One alternative is to send each port identified by the path query (see A1.4.2.1 “Query for Path” on page 1137) a DevMgtGet(ClassPortInfo) to determine if it is an IB managed I/O unit.

A1.5 STORAGE I/O

A1.5.1 IB STORAGE CONCEPTS

Storage is an I/O application where a storage initiator accesses a storage target (e.g., storage subsystem, storage adapter, or a storage device) via the IB fabric. This section addresses the considerations where the storage target takes the form of an I/O controller and the storage initiator (client of the storage controller) is instantiated as an IBA host or booting platform. There are several considerations in mapping a storage protocol onto IBA. That is, each protocol must specify values for protocol specific components in various management attributes.

An IBA storage controller should be sensitive to the richness of IBA’s scalable interconnect options. The controller may be deployed in direct attach, single host system, or multiple host system networks. Deployment of a storage controller in the latter may require host sharing capabilities in the controller.

Storage protocol is *the set of rules governing the content and exchange of storage related information passed between host platforms and I/O controllers of the InfiniBand Architecture*.

IBTA does not specify any one given storage protocol. External standards organizations define storage protocol mappings over IBA.

This section identifies the requirements of implementing specific storage protocols on IBA. It is expected that as storage protocols are mapped onto IBA by their respective standards bodies this section will grow to complement them.

A1.5.2 PROTOCOL SPECIFIC FIELDS

Certain components in IBA management attributes are defined to have protocol specific values. These components assist in the identification of the protocol supported by a given IOC, associate an IOC with a specific I/O device driver, or may aid in the usage of a protocol in a boot environment.

Values for the following components need to be defined by the organization specifying the protocol.

- IO Class
- IO Subclass
- Protocol
- Protocol Version
- ServiceName

Other management classes also have protocol specific components. This includes:

- AdditionalInfo component of BIS and Boot Management Locator Records (see Booting Annex)

A1.5.3 STORAGE PROTOCOLS

This section lists relevant storage protocols capable of operation over the IB fabric. These protocols have defined values for the IB components listed in section A1.5.2 “Protocol Specific Fields” on page 1138

- SCSI is a dominant block oriented storage protocol. SCSI RDMA Protocol (SRP) is a SCSI protocol capable of realizing the performance benefits of IBA’s RDMA semantics. Refer to T10/1415D, SCSI RDMA Protocol (SRP) standard for details.

ANNEX A2: CONSOLE SERVICE PROTOCOL

A2.1 INTRODUCTION

This annex is a supplement to Volume 1 of the InfiniBand Architecture specification, herein referred to as the base document. This annex specifies the IB Console Service Protocol.

A2.1.1 GLOSSARY

The following are additional terms not found in the Volume 1 Glossary (Chapter 2).

Console Display Object — A display terminal or window for displaying output from, and providing input to, a Console User.

Console Device — A Console Display Object.

Console IOC — An I/O controller that supports the Console Service Protocol and provides one or more Console Display Objects.

Console Server — An IOC or Host process that provides access to one or more Console Display Objects.

Console Server Process — A process on a host that supports the Console Service Protocol and provides one or more Console Display Objects.

Console User — One of a number of processes on a host that use the console service to input and output streams of console data. An example of a console user is a booting environment that uses the console service to display error messages on a console display.

CSP — Console Service Protocol

CSP Client — An entity that communicates with a CSP server (via the Console Service Protocol) to present virtual consoles to a set of console users.

CSP Connection — A connection that carries CSP messages.

CSP IOC — A Console IOC implementing the Console Service Protocol.

CSP Server — A Console Server implementing the Console Service Protocol.

CSP Session — A relationship between a Console User and a Console Display Object where data generated by the user is delivered to the display object and visa versa.

A2.1.2 COMPLIANCE

This annex specifies two new Compliance Categories (see Volume 1 Chapter 20 for explanation of compliance categories and qualifiers). These new categories are CSP Client and CSP Server.

There are two Compliance Qualifiers for CSP Server. They are CSP-IOC and CSP-Process.

Section A2.4 “Compliance Summary” on page 1171 provides a summary of compliance statements.

A2.1.3 OVERVIEW

Firmware and software on IBA hosts and IO Units often require console-based interaction with system administrators. Examples include the ROM-based boot environment, downloaded bootcode, OS loaders and IO controller firmware. This annex describes the IB Console Service Protocol (hereafter, CSP) that provides the *IB console service*. A variety of console strategies are possible; IBA supports the following.

- One or more *console display objects* (*e.g.* windows) behind a *CSP server process* on a host, where the CSP Server Process supports CSP.
- One or more *console devices* (*e.g.* terminals) behind a *CSP IOC* in an IOU, where the IOC supports the CSP.
- A *vendor-specific console abstraction* supported by one or more devices behind an IOC that supports a *proprietary protocol*. The IOU exports a downloadable expansion ROM image, which implements the vendor-specific console-device interface.

This section describes the *IB Console* abstraction and CSP, the wire protocol followed by IB console IOCs and IB console server processes. Booting with proprietary protocols is the subject of the Booting Annex; the vendor-specific console abstraction is outside the scope of IBA.

All multi-byte console protocol message fields use big endian byte ordering as described in section 1.5.1.

The difference between CSP IOCs and CSP server processes is the way by which the CSP client locates them. Locating the CSP IOC and the CSP server process is described in Section A2.2.1 on page 1144 and Section A2.2.2 on page 1145 and they are collectively referred to as *CSP servers*.

A2.1.4 GOALS

- Permit multiple console users within a host to use a single CSP connection, where each user has its own dedicated console display object.

- Support terminal emulation by a host process without the need for the host to dynamically create IOC GUIDs.
- Provide the conventional console-device abstraction of a pair of input and output devices.
- Provide for simple unformatted text input and output, as well as standard methods of inputting control characters and outputting simple formatted text.
- Expose a persistent device path for a console input/output device to firmware so it may be cached in nonvolatile memory and/or handed off from one console user to another.
- Allow for intelligence in a console device.

A2.2 THE IB CONSOLE ABSTRACTION

Figure 231 illustrates the overall architecture and operation of CSP. Typical users of IB console service (*console users*) are host-based execution threads, such as boot firmware, OS loader, system setup and diagnostic utilities, and loaded boot images.

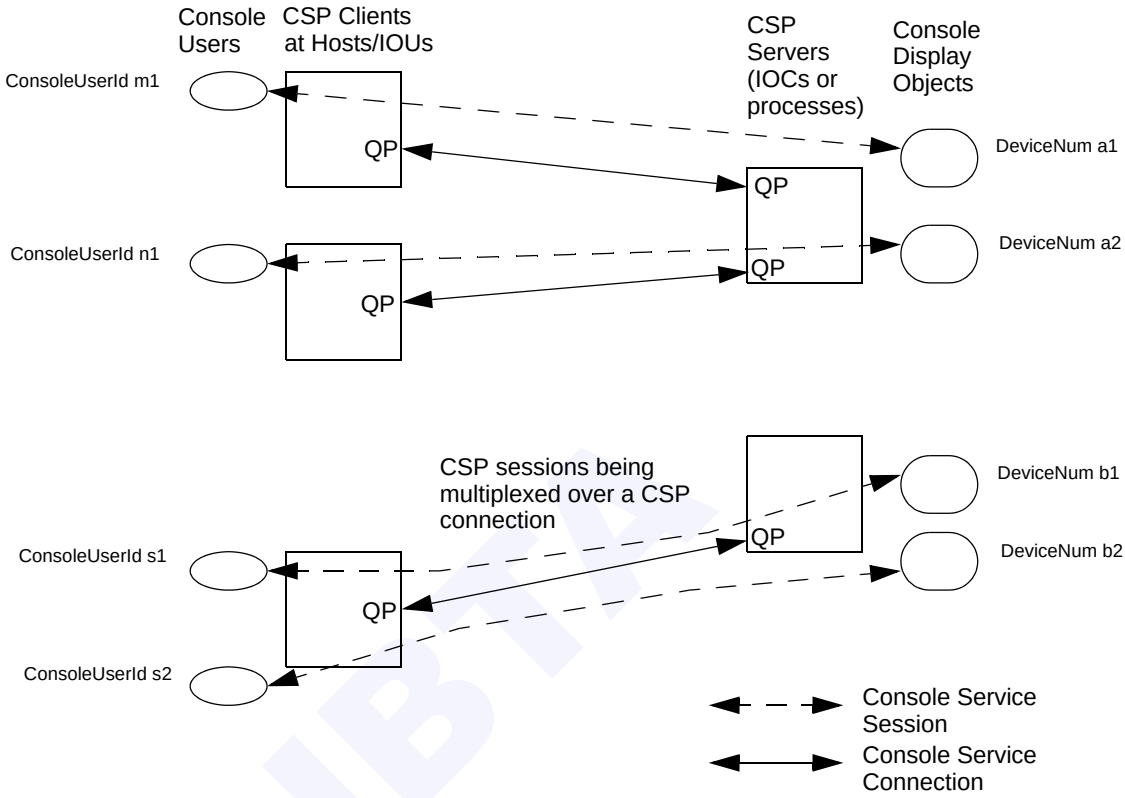


Figure 231 Schematic of IBA Console Abstraction

CA2-1: A CSP client shall assign a unique *console user ID* to each of its console users.

At each CSP server, there are one or more *console display objects* representing presentation interfaces.

CA2-2: A CSP server shall assign a unique *device number* to each console display object.

A CSP client establishes a *CSP connection* with a host or IOU CSP server by following the IBA CM protocol for establishing an RC channel.

CA2-3: A CSP server shall support RC service connections using only Send/Receive operations (i.e., does not use RDMA).

CA2-4: Every CSP message shall be sent with the SE bit in the BTH header set.

Whenever a CSP connection is needed, the CSP client shall be the one to initiate connection establishment.

A *CSP session* exists between a console user and a console display object. Multiple sessions may be multiplexed over a single CSP connection. Each console display object is associated with at most one console session.

The CSP client abstracts the console service as a CSP session to each of its console users. When multiple sessions are multiplexed over one CSP connection, it is the CSP client that demultiplexes between different console users sharing the connection.

Likewise, at the CSP server end, the server demultiplexes the traffic from a single CSP client with multiple sessions to different console display objects.

CA2-5: At any given time, a CSP server shall open at most one CSP session per console display object.

CSP sessions exists on top of a reliable connection service. A CSP session can survive the loss of a CSP connection caused, for instance, by the Boot environment to OS transition on a host or by a console-directed warm reboot. Session handles can be passed between console users during boot transitions, or saved by a dying console user and later recovered in order to resume a CSP session past a warm reboot. These properties of console service sessions are considered crucial to maintaining continuity in administrative interactions.

A2.2.1 CONSOLE IO CONTROLLERS

oA2-1: The CSP Server abstraction may be implemented by an I/O controller in a managed I/O unit, provided the IOC satisfies the following properties - refer to the I/O Annex and base document section 16.3 "Device Management".

- The IOC's Class, Subclass, Protocol, and Version values for its DevMgt IOControllerProfile attribute are as follows:
  - Class: 0x40FF
  - Subclass: 0xFFFF
  - Protocol: 0x0001
  - Version = 0x0001
- DevMgt *ServiceEntries*:*ServiceName* = "Console.IBTA"

- The service ID to be used by a CSP client in a console service connection request sent to such an IOC (hereafter, *CSP IOC*) can be found in DevMgt *ServiceEntries*:*ServiceID* for the Service Entry for that particular IOC with the matching ServiceName.

An IOU may contain multiple CSP IOCs, each offering console service behind a different service ID. Thus, a CSP IOC does not use the well-known Console ServiceID.

A CSP IOC supports CSP, as described below, for the enumeration, naming and control of console display objects.

A CSP IOC is not required to support multiple concurrent console service connections.

oA2-2: This compliance statement is obsolete.

The persistent identifier of a CSP IOC includes its IOC GUID and the Node or Port GUID of the channel adapter through which the IOC's services are accessed. If a CSP client wants to connect to the same display object, it must also retain the DeviceNum. A CSP client may retain these items of information in its persistent boot bindings.

A2.2.2 CONSOLE SERVER PROCESSES

The CSP Server abstraction may also be implemented by a host-based service offered by a *CSP server process* behind an HCA.

oA2-3: A CSP server process shall register its service with the SA by creating one or more ServiceRecords as follows. The ServiceName field shall contain the string "Console.IBTA". No service-specific or service-generic flags shall be set. The ServiceGID field shall contain the port GID of the CSP server process. The ServiceID shall be the well-known Console ServiceID 0x00000002. All other components are implementation specific.

Since CSP server processes can be started dynamically at any IBA host, console service should be considered both mobile and dynamic. The persistent identifier of a CSP server process contains just its port GUID and the well-known Console ServiceID. A CSP client may retain such information in its persistent boot bindings.

A2.3 CONSOLE SERVICE PROTOCOL

CSP is the message-level protocol used for establishing, maintaining, and communicating inside CSP sessions. The CSP Client uses CSP messages to establish a session between a console user and a console display object, modify the state of that session, and send and receive data streams over that session.

There are various ways for a CSP Client to discover the GID and Service ID of a CSP server (see Application Specific Identifiers Annex, Booting Annex, and Boot Information Service Annex). It uses this information to establish a reliable connection with the CSP Server.

CA2-6: CSP clients and CSP servers shall support CSP messages over IB RC transport service.

Once the CSP client establishes the connection, the general steps are:

- CSP Client sends ConsoleDeviceProfileRequest to learn console capabilities and establish buffer sizes. The CSP Server responds with the profile information
- For each Console User, the CSP Client sends a SessionInitRequest to establish (or reestablish) a session between a Console User and an Console Display Object. The CSP Server responds with an ACK (or NAK).
- The CSP Client sends Console Out data from the Console User and CSP Server sends Console In data from the Console Display object. The data is not interpreted by the CSP client or CSP Server, it is just passed to the Console Display Object and Console User.
- Either end can terminate a session. The CSP Client by sending a SessionEndRequest and the CSP Server by sending a SessionTerminated message.
- The CSP Client can also suspend sessions. Sessions that are suspended when the connection terminates are preserved.
- The CSP Client resumes a suspended session by sending a SessionResumemessage

The protocol takes place within the context of a CSP connection. A CSP session set up using CSP can be handed off from one console user to another without impacting the continuity of administrative interaction. Furthermore, a CSP session can be suspended before a host is rebooted, and later resumed after a new CSP connection has been established, allowing an administrator to continue monitoring a system through a warm reset with minimal loss of continuity.

Console I/O traffic from multiple sessions is multiplexed over a single console service connection. CSP session control messages are multiplexed with the console I/O traffic messages over that connection.

CSP session control messages are usually request-response messages. Requests originate from a CSP client and sent to a CSP server. Responses travel in the opposite direction. In most cases responses are generated as a result of a request message. Exceptions to this rule are

the SessionTerminated message (see Section 2.3.8 on page 1169) and Console I/O messages (see Section 2.3.5, "Normal Operation," on page 1159).

CA2-7: The CSP server shall send response messages for corresponding request messages on the same CSP connection on which the request message arrived.

Table 320 lists the CSP messages, indicates in which states those messages are valid, and indicates who can send the message. Figure 232 indicates the various session states, the messages or events that cause a state transition, and the messages that the CSP server sends in response to state transitions.

Table 320 Console Protocol Messages

Message Name	Op Code	Session State	Direction	Message Description
ConsoleDataOut	0	Active	From Client	Output from console user to display object
ConsoleDataIn	1	Active	To Client	Input from display object to console user
ConsoleDeviceProfileRequest	2	n/a	From Client	Request a Console Device Profile from the CSP server
ConsoleDeviceProfileReply	3	n/a	To Client	Response to ConsoleDeviceProfileRequest
ConsoleDeviceProfileReject	4	n/a	To Client	Response to ConsoleDeviceProfileRequest
SessionInitRequest	5	Idle	From Client	Establish a new session over a CSP connection
SessionInitAck	6	Transition to Active	To Client	Response to SessionInitRequest message (session granted)
SessionInitNAK	7	any	To Client	Response to SessionInitRequest message (request denied)
SessionSuspendRequest	8	Active	From Client	Suspend Request
SessionSuspendNAK	9	any	To Client	Suspend Reject
SessionSuspendAck	10	Transition to Suspend	To Client	Suspend Accept
SessionResumeRequest	11	Suspend	From Client	CSP SessionResumerequest

Table 320 Console Protocol Messages (Continued)

Message Name	Op Code	Session State	Direction	Message Description
SessionResumeAck	12	Transition to Active	To Client	CSP SessionResumerequest accepted
SessionResumeNAK	13	any	To Client	CSP SessionResumerequest declined
SessionEndRequest	14	Active or Suspend	From Client	Request to terminate CSP session
SessionTerminated	15	Transition to Idle	To Client	Accept SessionEndRequest request or initiate termination
PingRequest	16	Active	Either	Request the other side to respond with a PingResponse within the PingResponse-Time negotiated when the session was started.
PingResponse	17	Active	Either	Response to PingRequest
ErrorReport	18	any	Either	Reports protocol error causes

CA2-8: A CSP Client shall not send a message that has a direction as specified in Table 320 of "To Client".

CA2-9: A CSP Server shall not send a message that has a direction as specified in Table 320 of "From Client".

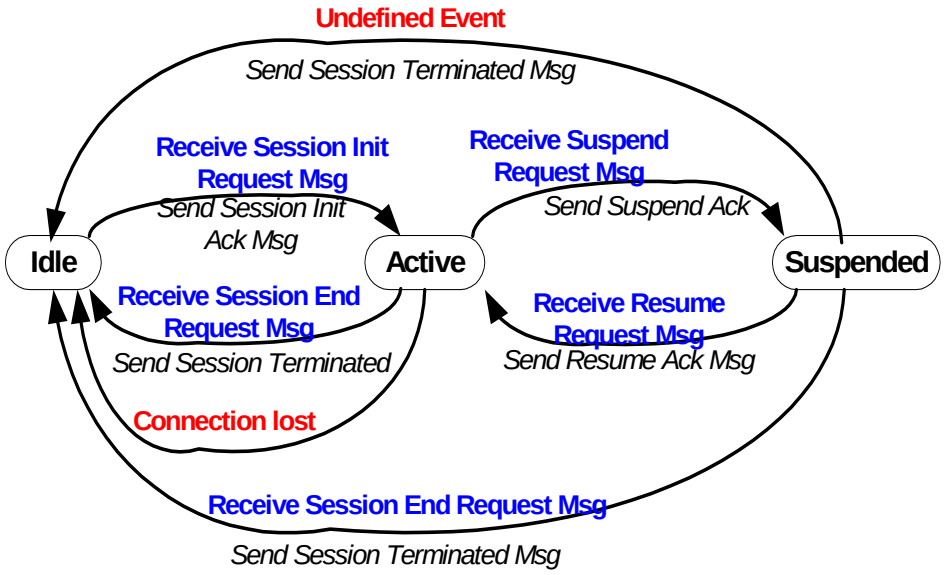


Figure 232 CSP Server Session State Diagram

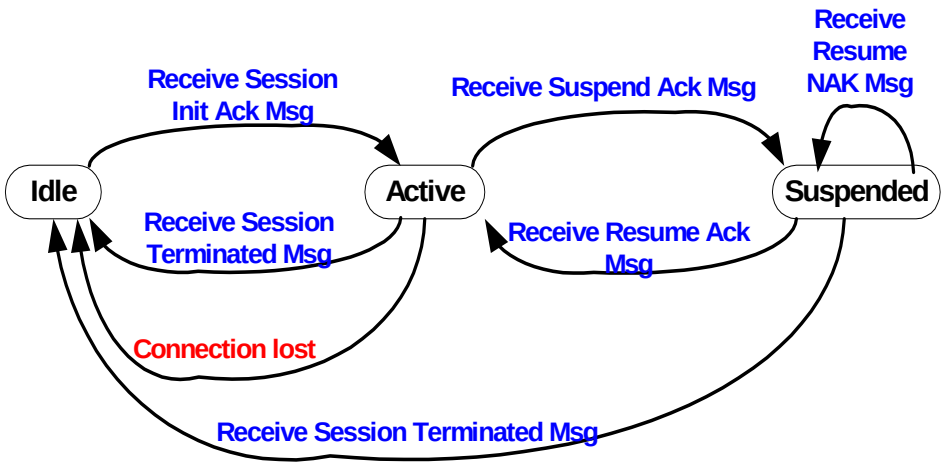


Figure 233 CSP Client Session State Diagram

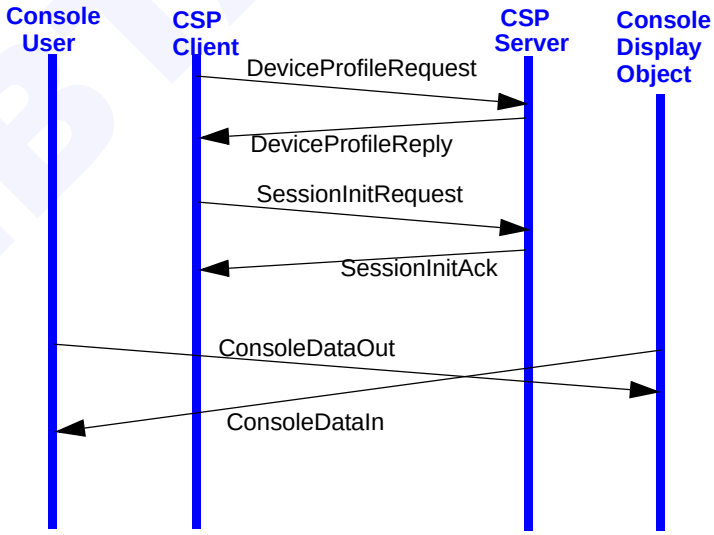


Figure 234 Protocol Flow Diagram

Table 321 specifies actions that the CSP client and CSP server take if they receive an invalid message.

CA2-10: The CSP client and CSP server shall respond to protocol errors as per Table 321

Table 321 Console Protocol Error Actions

Error	Session State	Direction	Reaction
Inappropriate or Invalid OpCode	any	either	Reject message via an ErrorReport message
Invalid Session (Device Number + ConsoleUserId)	any	To Server	Respond with SessionTerminated message
Invalid Session (Device Number + ConsoleUserId)	any	To Client	Discard message - May optionally send SessionEndRequest message
Specified Device Number not available	any	To Server	• Reject a ConsoleDeviceProfileRequest message via a ConsoleDeviceProfileReject message, • reject a SessionInitRequest message via a SessionInitNAK message, • reject any other message via a SessionTerminated message
Invalid ErrorReport Message	any	either	Discard message - do no respond
Any other invalid component value	any	either	Reject message via an ErrorReport message
Message too short	any	either	Reject message via an ErrorReport message
Message too long	any	either	Reject message via an ErrorReport message

Neither the CSP client nor the CSP server check the content of the Data components in ConsoleDataIn and ConsoleDataOut messages. How Console Users and Console Display Objects respond to errors in the data stream are outside the scope of the Console Service Protocol.

A2.3.1 ERROR REPORTING

The CSP client or CSP server send an ErrorReport when it detects an invalid component value in a message that prevents it from properly processing the received message as per Table 321

There is no response to an ErrorReport message. Receiving an error report indicates a protocol error. The overall effect could be that either end might terminate the session or the connection.

Table 322 ErrorReport

Component Name	Offset (bytes)	Length (bytes)	Component Description
CSPOpCode	0	4	0xFFFFFFFF: ErrorReport message
DeviceNum	4	4	DeviceNum value from offending message
ConsoleUserId	8	4	ConsoleUserId value from offending message

Table 322 ErrorReport (Continued)

Component Name	Offset (bytes)	Length (bytes)	Component Description
OpCode	12	4	CSPOpCode from offending message.
RejectCode	16	2	• 0x0001 = Invalid component value (ComponentOffset indicates component with bad value) • 0x0002 = Message too short (ComponentOffset indicates expected length in bytes) • 0x0003 = Message too long (ComponentOffset indicates expected length in bytes) • 0xFFFF = any other protocol error all other values reserved
reserved1	18	2	reserved
ComponentOffset	20	4	Location of the offending component in the offending message expressed as the offset in number of bytes from the CSPOpCode (i.e., a value of zero indicates the CSPOpcode, a value of 4 indicates the DeviceNum component, etc.)

A2.3.2 CONSOLE DEVICE ENUMERATION

IB console capabilities are enumerated in a *ConsoleDeviceProfile* structure (see Table 323 on page 1151). It provides sufficient information for configuring the console-specific parameters of standard booting environments (*e.g.*, Intel Corporation's Extended Firmware Interface (EFI) and the IEEE 1275-1994 Open Firmware standard).

The Console Device Profile carries a number of Console Capability records (one for each capability type). The CSP Server provides this information to the CSP Client in the ConsoleDeviceProfileReply message. The only standard capability type is "Terminal Type", but the protocol allows for proprietary capabilities. The CSP Client selects the appropriate capability by providing a Console Capability Record in SessionInitRequest and SessionResumeRequest messages (CapabilitiesUsed component).

Table 323 ConsoleDeviceProfile

Component Name	Offset (bytes)	Length (bytes)	Component Description
MaxUsers	0	2	The maximum number of sessions allowed per console service connection
CapabilityCount	2	2	The number of capability listings (max. 255)

Table 323 ConsoleDeviceProfile (Continued)

Component Name	Offset (bytes)	Length	Component Description
CapabilityList	4	n x 12	12 bytes per capability record, as specified in Table 324, where n is the value of CapabilityCount

Table 324 ConsoleCapabilityRecord

Component Name	Offset (bytes)	Length (bytes)	Component Description
CapabilityType	0	4	0: Undefined, 1: Proprietary (not applicable to console server processes), 2: Terminal type All other values: reserved
ContentFormat	4	8	Not relevant for CapabilityType = Undefined. For CapabilityType = Proprietary, interpretation of this field is specific to Subsystem VendorID and SubsystemID fields of the CSP IOC's ControllerProfile. For CapabilityType = Terminal type, a bitmask describing the content formats supported by the console device, as follows. Byte:Bit position, Corresponding content format 0:0, ASCII input and output (**always set** in ConsoleDeviceProfileReply messages (Table 326 on page 1155); required capability). ISO 646 defines the character set and control character interpretation for the ASCII character set. 0:1, UTF-8, ISO 10646, input and output 0:2, HTTP over CSP Session (HTTP/1.1 GET and POST messages only; see Section 2.3.5.3 on page 1160.) all other bits: reserved In ConsoleDeviceProfileReply messages (Table 326 on page 1155), multiple bits of this field may be set in order to communicate the full list of content formats supported by the CSP server. In SessionInitRequest messages (Table 328 on page 1157), however, no more than one bit may be set because it specifies the format selected by the CSP client for use during that session. (see CA2-29: on page 1158)

CA2-11: ContentFormat bit 0 (ASCII Input and output) shall always be set in Console Capability Records.

CA2-12: If the ContentFormat bit 1 (UTF-8) of a Console Capability Record is set to 1 by a CSP Server, then it shall support ISO 10646 defined UTF-8 data streams.

CA2-13: If the ContentFormat bit 2 (HTTP) of a Console Capability Record is set to 1 by a CSP Server, then it shall support HTTP/1.1 GET and POST messages.

CA2-14: Fields that are reserved shall be set to zero in messages being sent and ignored in received messages.

CA2-15: Bits that are reserved shall be set to zero in messages being sent and ignored in received messages.

CA2-16: Values that are reserved shall not be used in messages being sent and treated as unknown in received messages. Receiving a reserved value is not necessarily an error, but may be treated as an error.

A2.3.3 CAPABILITY QUERY

A CSP client conducts capability negotiation with a CSP server on behalf of a console user. The first step is to query the CSP Server for the capability of it's console display objects using the *ConsoleDeviceProfileRequest* message. The CSP server shall use the ConsoleUserId (see Table 326 on page 1155) in generating its response. It should be noted that no device numbers are granted or reserved by the CSP server as a result of capability query. This set of messages are also used to establish buffer sizes and negotiate version level.

CA2-17: A CSP server shall support receiving a *ConsoleDeviceProfileRequest* message (described in Table 325 on page 1154) over a CSP connection at any time during the connection's lifetime.

CA2-18: A CSP server shall reply to a *ConsoleDeviceProfileRequest* message with a *ConsoleDeviceProfileReply* message (described in Table 326 on page 1155) containing the requested *ConsoleDeviceProfile* (described in Table 323 on page 1151 and Table 324 on page 1152).

It is not required that the same values be returned in response to repeated *ConsoleDeviceProfileRequest* messages.

These communications take place within the context of a CSP connection, but outside any CSP session context.

Traffic is asynchronous in both directions and thus both ends need to have receive buffers posted at all times. The client and server specify buffer size in the ConsoleDeviceProfileRequest and Reply.

CA2-19: Neither the client nor the server shall reduce its buffer size while there are active console sessions.

Table 325 ConsoleDeviceProfileRequest

Component Name	Offset (bytes)	Length (bytes)	Component Description
CSPOpCode	0	4	0x00000002: Request a Console Device Profile from a CSP server
DeviceNum	4	4	0: send capabilities available for a new session over this CSP connection Non-zero: used in conjunction with ConsoleUserId and CSP client information to determine the capabilities available for reconnecting to the display object identified by DeviceNum
ConsoleUserId	8	4	Unique identifier of the console user at the CSP client
Version	12	2	The highest version number of CSP messages supported by the CSP client sending this request. Byte 0: Major version Byte 1: Minor version This document specifies version 0x0100.
reserved	14	2	reserved
ClientBufferSize	16	4	Specifies the maximum number of bytes of a message that the CSP server can send to the CSP client. The minimum value shall be 256.
PingResponseTime	20	2	Maximum number of 10 milliseconds time periods this CSP client will take to respond to a PingRequest message with a PingResponse.

CA2-20: If the CSP server supports the version in the request, it shall return ConsoleDeviceProfileReply with that version number.

CA2-21: If the CSP server supports a lower version than in the request, it shall return ConsoleDeviceProfileReply with the highest version that it supports.

CA2-22: If the CSP server supports only higher versions than requested, it shall return ConsoleDeviceProfileReject with the lowest version it supports.

CA2-23: If the CSP server supports versions below the requested version, but not the version requested, it shall return ConsoleDeviceProfileReplyReject with the highest version it supports below the version requested.

CA2-24: The CSP client shall either use the version returned, submit another ConsoleDeviceProfileRequest, or shall terminate the connection

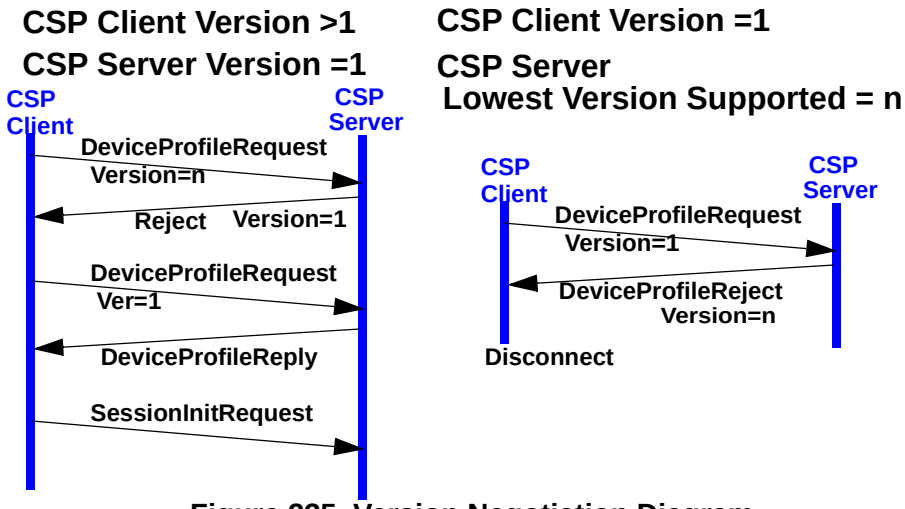


Figure 235 Version Negotiation Diagram

Table 326 ConsoleDeviceProfileReply

Component Name	Offset (Bytes)	Length (Bytes)	Component Description
CSPOpCode	0	4	0x00000003: CSP Server response to ConsoleDeviceProfileRequest from a CSP client
DeviceNum	4	4	From the request
ConsoleUserId	8	4	From the request
Version	12	2	The highest version number of CSP messages supported by the CSP server sending this reply. Byte 0: Major version Byte 1: Minor version This document specifies version 0x0100.
reserved	14	2	
ServerBufferSize	16	4	Specifies the maximum number of bytes of a message that the CSP client can send to the CSP server. The minimum value shall be set to 256.
PingResponseTime	20	2	Maximum number of 10 milliseconds time periods this CSP server will take to respond to a PingRequest message with a PingResponse.
MaxUsers	22	2	The maximum number of active+suspended sessions allowed per console service connection.
reserved	24	2	
CapabilityCount	26	2	The number of capability records (max. 255)
CapabilityList	28	n x 12	12 bytes per capability record, as specified in Table 324 on page 1152, where n is the value of Capability count. *n* shall be zero if the ConsoleDeviceProfileRequest message contained an invalid DeviceNum.

Table 327 ConsoleDeviceProfileReject

Component Name	Offset (bytes)	Length (bytes)	Component Description
CSPOpCode	0	4	0x00000004: Reject service
DeviceNum	4	4	From the request
ConsoleUserId	8	4	From the request
Version	12	2	If the requested CSP version is lower than the lowest version supported by the server, reject with the lowest CSP version supported by the server. If the requested version is an unsupported version between the lowest and highest versions supported by the server, reject the highest version supported by the CSP server that is less than the requested version. Byte 0: Major version Byte 1: Minor version
Reserved	14	2	Reserved
Reject Code	16	2	0x0001 = Can not support version 0x0002 = DeviceNum not active or not valid Others reserved

A2.3.4 SESSION ESTABLISHMENT

CSP session establishment protocol runs outside of the context of any specific session but within the context of a CSP connection.

Capability negotiation (see Section 2.3.3 on page 1153) is recommended but optional before a console service session is established or re-established.

A console user may optionally issue a *ConsoleDeviceProfileRequest* message to its CSP server before sending a *SessionInitRequest* message.

To establish a new CSP session a CSP client shall send a *SessionInitRequest* message to the CSP server.

Table 328 SessionInitRequest

Component Name	Offset (bytes)	Length (bytes)	Component Description
CSPOpcode	0	4	0x00000005: Establish a new session over a CSP connection
DeviceNum	4	4	Requested deviceNum if known, else set to zero to mean any.
ConsoleUserId	8	4	Unique identifier of the console user at the CSP client requesting a new session
Nickname	12	8	Short, null-terminated UTF-8 ASCII string identifying the console user
ConsoleUserName	20	64	Long, null-terminated UTF-8 ASCII string identifying the console user
CapabilitiesUsed	84	12	The Console Capability Record (see Table 324) selected by the requesting console user for use during the session being established

The Nickname and ConsoleUserName components provide a means for the console user to identify itself. The exact application of this information is not specified. However, the expectation is that console server with multiple windows would use the ConsoleUserName as the window title and that the Nickname might proceed each line on a console display for when a single physical display is used for multiple console display objects.

Table 329 SessionInitAck

Component Name	Offset (bytes)	Length (bytes)	Component Description
CSPOpcode	0	4	0x00000006: Response to SessionInitRequest message (session granted)
DeviceNum	4	4	A non-zero device number assigned to the console display object allocated to the session being established.
ConsoleUserId	8	4	From request
ReferenceCount	12	4	The count of active sessions on the CSP connection on which the response is being sent, including the one just established

CA2-25: A CSP server shall not assign a DeviceNum of zero.

Table 330 SessionInitNAK

Component Name	Offset (bytes)	Length (bytes)	Component Description
CSPOpcode	0	4	0x00000007: Response to SessionInitRequest message (request denied)
DeviceNum	4	4	Reserved
ConsoleUserId	8	4	From request
ReferenceCount	12	4	The count of currently active sessions over the CSP connection on which the response is being sent
NAKCode	16	4	0: reserved 1: DeviceNum already in use or invalid 2: Requested capability not available 3: MaxUsers exceeded for console service connection 4: All devices in use 5: ConsoleUserId in use or not available 6: Not otherwise specified 0xFF80-0xFFFF: Vendor specific error (interpretation is console specific and unless otherwise known, treat as "device not available"). All other values: reserved

CA2-26: A CSP server shall respond to a SessionInitRequest message (Table 328 on page 1157) by either establishing a CSP session and sending a SessionInitAck message (Table 329 on page 1157) or by rejecting the session-establishment request, i.e., by sending a SessionInitNAK message (Table 330 on page 1158).

CA2-27: A CSP client shall include the device number — returned by the CSP server in the *SessionInitAck* message — in all subsequent messages for that session.

CA2-28: All console messages for a CSP session shall contain that session's ConsoleUserId and DeviceNum.

CA2-29: The CSP client shall set exactly one bit of the ContentFormat field in a SessionInitRequest message (Table 328 on page 1157), for CapabilityType = Terminal Type.

A2.3.5 NORMAL OPERATION

The following messages transfer console I/O traffic within the context of a CSP session.

Table 331 ConsoleDataOut

Component Name	Offset (bytes)	Length (bytes)	Component Description
CSPOpcode	0	4	0x00000000: Output from console user to display object
DeviceNum	4	4	Indicates device number assigned to the associated console display object
ConsoleUserId	8	4	Indicates the console user
Length	12	4	Length of input or output data (max 4080)
Data	16	as specified by the Length component	Opaque contents of the console I/O protocol between a console user and a console display object. Contents must follow selected content format specified in the selected Console Capability Record.

Table 332 ConsoleDataIn

Component Name	Offset (bytes)	Length (bytes)	Component Description
CSPOpcode	0	4	0x00000001: Input from display object to console user
DeviceNum	4	4	Indicates device number assigned to the associated console display object
ConsoleUserId	8	4	Indicates the console user
Length	12	4	Length of input or output data (max 4080)
Data	16	as specified by the Length component	Opaque contents of the console I/O protocol between a console user and a console display object. Contents must follow selected content format specified in the selected Console Capability Record.

CA2-30: ConsoleDataIn messages shall travel only from a CSP server to a CSP client.

CA2-31: ConsoleDataOut messages shall travel only from a CSP client to a CSP server.

CA2-32: If a session is established for ASCII, then the ASCII byte stream shall conform to ISO 646.

CA2-33: If a session is established for UTF-8, then the byte stream shall conform to ISO/IEC 10646

A2.3.5.1 ASCII TEXT STREAMS

A CSP client establishes an ASCII session using the normal session establishment messages described in Section 2.3.4 on page 1156. The SessionInitRequest must set Terminal Type:ASCII bit in the CapabilitiesUsed component and the message must be acknowledged by the CSP server with a SessionInitAck message.

The content of ConsoleDataIn and ConsoleDataOut messages are not interpreted by the CSP client nor CSP server. Rather the content is passed between the Console User and the Console Display Object. These messages do not imply any framing (i.e., no implied CR/LF characters) and thus, long text strings can be passed using multiple messages.

A2.3.5.2 UTF-8 TEXT STREAMS

A CSP client establishes a UTF-8 session using the normal session establishment messages described in Section 2.3.4 on page 1156. The SessionInitRequest must set Terminal Type:UTF-8 bit in the CapabilitiesUsed component and the message must be acknowledged by the CSP server with a SessionInitAck message.

The content of ConsoleDataIn and ConsoleDataOut messages are not interpreted by the CSP client nor CSP server. Rather the content is passed between the Console User and the Console Display Object. These messages do not imply any framing (i.e., no implied CR/LF characters) and thus, long text strings can be passed using multiple messages.

A2.3.5.3 HTTP CONSOLE SUPPORT

HTML pages or other HTTP-supported content may be transferred from a console user to a console display object using the HTTP/1.1 protocol[27]. In order to accomplish this, a CSP session must be properly set up to serve as HTTP/1.1 transport. Such a session is called an *HTTP-over-CSP session*.

The CSP client establishes an HTTP-over-CSP session using the normal session establishment messages described in Section 2.3.4 on page 1156. The CSP client sends the SessionInitRequest setting Terminal Type of HTTP-over-CSP bit in the CapabilitiesUsed component and the message must be acknowledged by the CSP server with a SessionInitAck message.

Once the session is established, the console user shall assume the role of an HTTP/1.1 server supporting at least the GET and POST methods. The console display object shall assume the role of an HTTP/1.1 client and use only the GET or POST methods. Both shall accept HTTP/1.1 chunked transfer encoding (see RFC 2616[27] for details). Both should use

27. Hypertext Transfer Protocol -- HTTP/1.1, RFC 2616, IETF, June 1999.

the *Connection: Keep-Alive* option in HTTP request and response headers.

All HTTP/1.1 messages (requests as well as responses) shall be fragmented into chunks of size 4080 bytes or less using chunked transfer encoding. All HTTP/1.1 messages, or chunks thereof, shall be carried in the payload area (Data field) of Console Input and Console Output messages. The console user and the console display object shall be responsible for all fragmentation and reassembly required to fit the messages within the 4080-byte payload limit.

HTTP connections are closed by terminating the console session.

A2.3.6 SESSION HANDOFF AND MAINTENANCE

Session maintenance allows a console user to survive a connection change and it also allows a session to be handed-off to a new user. In the first case, a console user may suspend a session before it loses its CSP connection. It then resumes the session after a new connection has been established, for instance after a warm reboot. In the second case, a console user hands off its session to a different console user, for example during a boot transition.

Session maintenance messages allow a returning console user to resume a session without losing continuity of console interaction and without a requirement for a CSP client to renegotiate console capabilities with a CSP server.

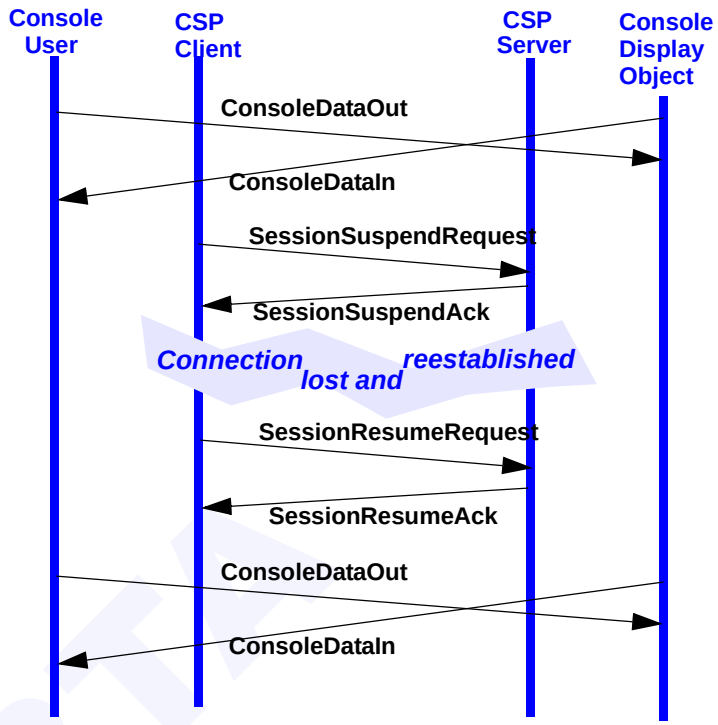


Figure 236 Suspend Example

A session resume request with the originally negotiated console capabilities, submitted within the SuspendTime returned in the SessionSuspendAck (Table 335 on page 1166) message, is expected to succeed. Only in cases of a crash or reconfiguration of the CSP server does the resume request fail.

A session may be resumed with new capabilities and it is possible that the resume request will not be honored if the CSP server does not support the new capabilities. There may also be resume failures after a crash or reconfiguration of a CSP server.

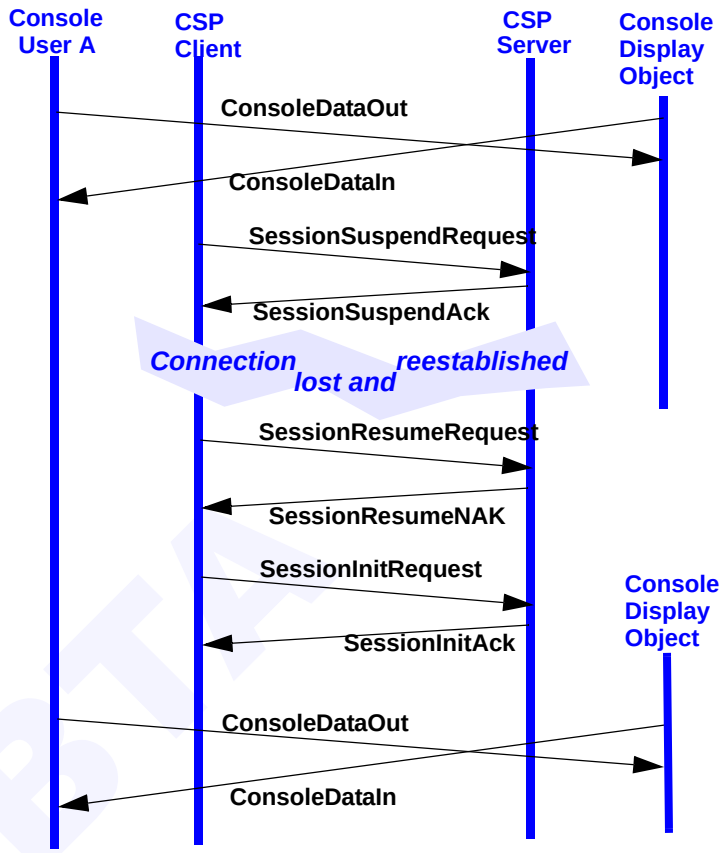


Figure 237 Resume Reject

An example of a resume request with new capabilities is a boot platform that runs with ASCII capabilities and the booted system tries to resume the session with HTTP capabilities on a new RC. If the CSP server does not support the new capabilities, it will reject the resume request.

Handoff is similar to session resumption except that a new console user inherits the session and capabilities must be renegotiated. The console display object supporting the session may flush interactive streams and reset display state, for instance, if new capabilities are negotiated.

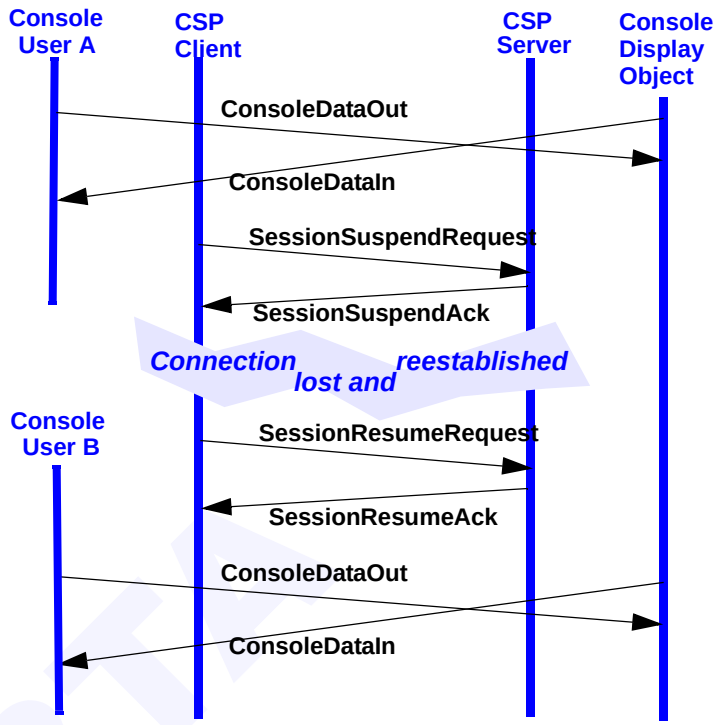


Figure 238 Handoff Example

CA2-34: A CSP server shall support CSP sessions being handed off between two console users over the same CSP connection using messages described in Table 333, Table 334, Table 335, Table 336, Table 337, and Table 338.

CA2-35: A CSP server shall support a CSP session being suspended and subsequently resumed over a new CSP connection using messages described in Table 333, Table 334, Table 335, Table 336, Table 337, and Table 338.

CA2-36: A CSP server that accepts a session maintenance request shall quiesce its end of the session and go into a state where it expects an optional *ConsoleDeviceProfileRequest* message followed by a *SessionResumeRequest* message from the resuming console user indicating the current device number.

The message formats are as follows:

Table 333 SessionSuspendRequest

Component Name	Offset (bytes)	Length (bytes)	Component Description
CSPOpcode	0	4	0x00000008: Suspend Request
DeviceNum	4	4	Indicates device number assigned to the associated console display object
ConsoleUserId	8	4	Indicates previous console user
NewConsoleUserId	12	4	Indicates new console user
SuspendTime	16	4	32 bit unsigned integer indicating in seconds the duration for which the resources associated with the session are requested to be preserved by the CSP server.

Table 334 SessionSuspendNAK

Component Name	Offset (bytes)	Length (bytes)	Component Description
CSPOpcode	0	4	0x00000009: Suspend Reject
DeviceNum	4	4	From request
ConsoleUserId	8	4	From request
ReferenceCount	12	4	The count of active sessions on the CSP connection on which the response is being sent
NAKCode	16	4	0: Out of resources 1: DeviceNum is invalid 2: ConsoleUserId is invalid 0xFF80-0xFFFF: Other error (interpretation is console specific) All other values: reserved

Table 335 SessionSuspendAck

Component Name	Offset (bytes)	Length (bytes)	Component Description
CSPOpcode	0	4	0x0000000A: Suspend Accept
DeviceNum	4	4	Device number of the console display object which is being put in suspended mode (from request)
ConsoleUserId	8	4	ConsoleUserId of the new console user (from request)
ReferenceCount	12	4	The count of active sessions on the CSP connection on which the response is being sent
ResumeKey	16	8	64-bit key generated by the CSP server for the associated device number and requesting console user for validation of session resumption request. The server expects this key to be present in a SessionResumerequest for the device number in question (see Table 336 on page 1166)
SuspendTime	24	4	32 bit unsigned integer indicating in seconds the duration for which the resources associated with the session are requested to be preserved by the CSP server. This value will be less than or equal to the SuspendTime value in the SessionSuspendRequest.

Table 336 SessionResumeRequest

Component Name	Offset (bytes)	Length (bytes)	Component Description
CSPOpcode	0	4	0x0000000B: CSP SessionResumerequest
DeviceNum	4	4	Device number of the console display object with which to resume session
ConsoleUserId	8	4	ConsoleUserId of the requesting console user
Reserved1	12	4	reserved
ResumeKey	16	8	64-bit key generated by the CSP server at the time of accepting session maintenance request corresponding to this resume request (see Table 335 on page 1166)
Nickname	24	8	Short, null-terminated UTF-8 ASCII string identifying the new console user
ConsoleUserName	32	64	Long, null-terminated UTF-8 ASCII string identifying the new console user
CapabilitiesUsed	96	12	The Console Capability Record (see Table 324) selected by the requesting console user for use during the session being resumed

On a successful session suspend, the CSP Server retains the state information of the session including the ResumeKey. A SessionResumeRequest with the correct ResumeKey can modify certain parameters.

The SessionResumeRequest allows the CSP client to specify a new ConsoleUserId for flexibility. This may be useful in the case where the boot environment hands off to the OS and the OS has different Console UserIDs from the boot environment, but wants to use the same console display object.

Table 337 SessionResumeAck

Component Name	Offset (bytes)	Length (bytes)	Component Description
CSPOpcode	0	4	0x0000000C: CSP SessionResumeRequest accepted
DeviceNum	4	4	From request
ConsoleUserId	8	4	From request
ReferenceCount	12	4	The count of active sessions on the CSP connection on which the response is being sent, including the one just resumed

Table 338 SessionResumeNAK

Component Name	Offset (bytes)	Length (bytes)	Component Description
CSPOpcode	0	4	0x0000000D: CSP SessionResumerequest declined
DeviceNum	4	4	From request
ConsoleUserId	8	4	From request
ReferenceCount	12	4	The count of currently active sessions over the CSP connection on which the response is being sent
NAKCode	16	4	• 0: Out of session resources • 1: There is an active session on the requested DeviceNum • 2: A requested capability in the CapabilitiesUsed field of the request is no longer available. • 3: Invalid DeviceNum in request • 4: There is a suspended CSP session on the specified DeviceNum but the request did not contain the matching ConsoleUserId • 5: ResumeKey mismatch for the specified DeviceNum • 0xFF80-0xFFFF: Other error (interpretation is console specific) All other values: reserved

A2.3.7 CONNECTION MAINTENANCE MESSAGES

The PingRequest and PingResponse messages allow a CSP client or a CSP server to verify that the function on the other end of the RC connection is still responding to requests.

Standard RC management interfaces, CM DisconnectRequest and RNR Nak for example, also indicate problems on the RC.

Table 339 PingRequest

Component Name	Offset (bytes)	Length (bytes)	Component Description
CSPOpcode	0	4	0x00000010: CSP PingRequest

Table 340 PingResponse

Component Name	Offset (bytes)	Length (bytes)	Component Description
CSPOpcode	0	4	0x00000011: CSP PingResponse

CA2-37: A CSP client or server that receives a PingRequest message shall respond with a PingResponse message within it's PingResponse-Time period.

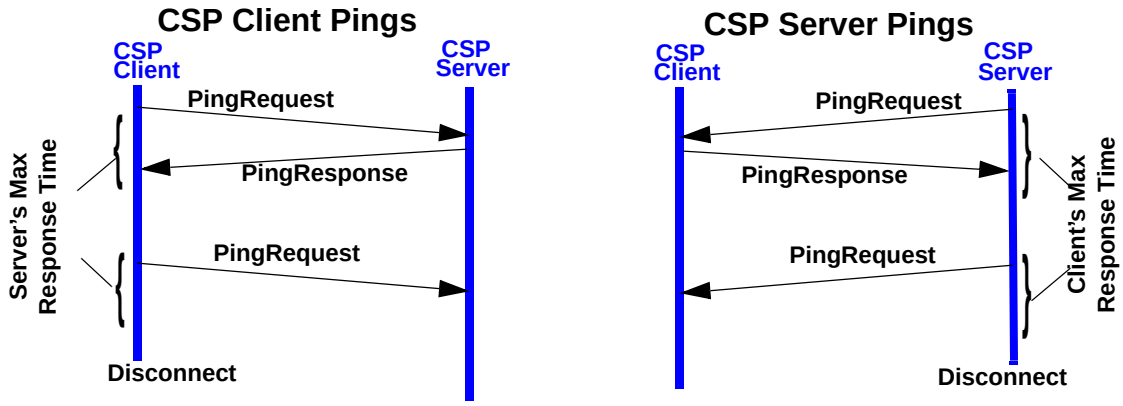


Figure 239 Ping Diagrams

The CSP client PingResponseTime period is specified in the data included in his ConsoleDeviceProfileRequest. The CSP server PingResponseTime period is specified in the data returned in his ConsoleDeviceProfileResponse.

If a CSP client does not receive a PingResponse message within the expected timeout period, it is allowed to tear down the RC session and notify affected Console Users that the session had been terminated.

If a CSP server does not receive a PingResponse message within the expected timeout period, it is allowed to transition affected Console Display Objects to the idle state.

A2.3.8 SERVICE CONNECTION AND SESSION TERMINATION

Normal session tear-down may be initiated either by a CSP client or by a CSP server.

Table 341 SessionEndRequest

Component Name	Offset (bytes)	Length (bytes)	Component Description
CSPOpcode	0	4	0x0000000E: Request to terminate CSP session
DeviceNum	4	4	Device number of the console display object associated with the CSP session to be terminated
ConsoleUserId	8	4	Console User ID of the console user associated with the CSP session

Table 342 SessionTerminated Message

Component Name	Offset (bytes)	Length (bytes)	Component Description
CSPOpcode	0	4	0x0000000F: Accept termination request
DeviceNum	4	4	From SessionInitAck message or from SessionEndRequest message
ConsoleUserId	8	4	From SessionInitRequest message or from SessionEndRequest message
ReferenceCount	12	4	Count of remaining active CSP sessions over this service connection following termination of this session

CA2-38: A CSP server shall issue a SessionTerminated message (Table 341 on page 1169) upon receiving a SessionEndRequest message (Table 342 on page 1169).

A CSP server may optionally issue a SessionTerminated message to the CSP client in order to unilaterally terminate a CSP session.

CA2-39: A CSP server shall issue a SessionTerminated message on receiving a ConsoleDataOut message not associated with any active CSP session.

When a CSP client receives a SessionTerminated message (Table 342 on page 1169) with ReferenceCount set to zero, it may close the CSP connection immediately.

The CSP server may close the CSP connection after completion of the send of SessionTerminated message (Table 342 on page 1169) with ReferenceCount set to zero.

Any sessions not suspended using SessionSuspendmessages are considered terminated when the underlying CSP connection is closed.

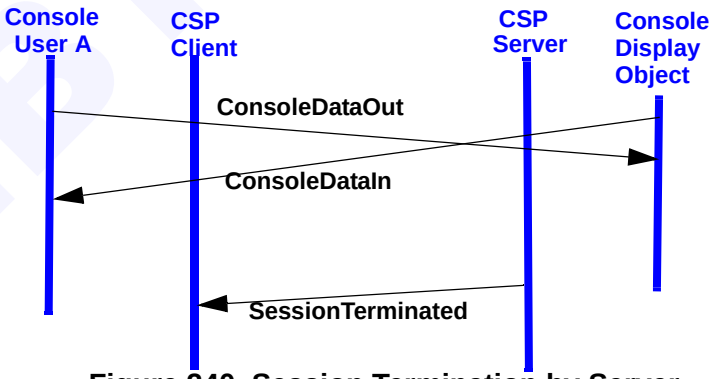


Figure 240 Session Termination by Server

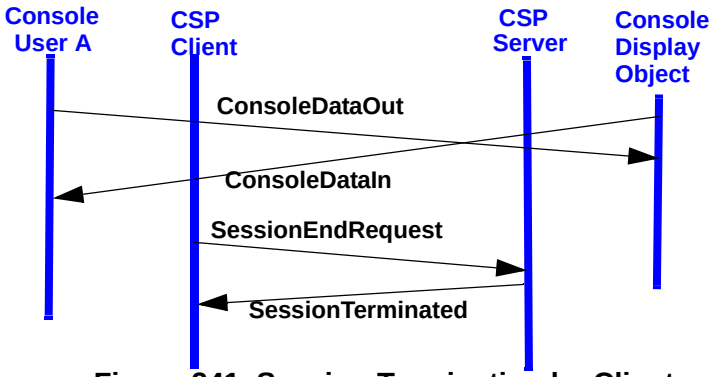


Figure 241 Session Termination by Client

A2.4 COMPLIANCE SUMMARY

This annex specifies two new Compliance Categories (see Volume 1 Chapter 20 for explanation of compliance categories and qualifiers). These new categories are:

- CSP client
- CSP server.

There are no Compliance Qualifiers for CSP client.

A2.4.1 CSP CLIENT COMPLIANCE CATEGORY

In order to claim compliance to the InfiniBand Architecture Specification for the Compliance Category of Console-Client, a product shall meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support (currently there are no optional compliance qualifiers for this category).

A2.4.2 CSP SERVER COMPLIANCE CATEGORY

In order to claim compliance to the InfiniBand Architecture Specification for the Compliance Category of Console-Server, a product shall meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support.

There are two Compliance Qualifiers for CSP Server. They are:

- CSP-IOC - an I/O controller providing console service
- CSP-Process - a console server process running on a host providing console service

ANNEX A3: APPLICATION SPECIFIC IDENTIFIERS

A3.1 INTRODUCTION

This document is a supplement to Volume 1 of the InfiniBand Architecture, herein referred to as the base document. This document provides an annex to the base document that specifies application specific values for InfiniBand management components and policies for using those components.

The main topics are:

A3.2 Service ID - Addresses provisions and policies for supporting services over the InfiniBand fabric. Specifically policies for assigning and using Service IDs.

A3.3 I/O Controller Identification - Specifies policies for creating and interpreting vendor and protocol specific values in Device Management class IOControllerProfile attributes

A3.4 Service Names - Specifies policies for creating Service Names used in various management attributes.

A3.5 Management Class Codes - Lists application specific management classes.

A3.1.1 GLOSSARY

The following are additional terms not found in the Volume 1 Glossary (Chapter 2).

Platform — A logical node that consists of one or more channel adapters under the control of a single operating system instance.

Service — A service is a function, or set of functions, accessible by a known protocol (i.e., the service protocol). A service is identified by its Service Name.

Service Client — The user of a service. Typically, the service client contacts the service provider and sends it service requests

Service GID — A GID that is used to obtain a path to a service. A service provider makes known its service GID and a service client uses that GID to obtain a path to the service provider.

Service Name — A name used to identify a service. Service provides usually register their services by service name and service clients use the service name to locate the service.

Service Provider — An entity that provides a service. Typically a service provider waits for service clients to contact it and responds to service requests.

A3.1.2 COMPLIANCE

This annex does not define functionality, but rather defines the policies for administering certain InfiniBand values.

This annex specifies 3 new Compliance Categories (see Volume 1 Chapter 20 for explanation of compliance categories and qualifiers). These new categories are:

- Service ID Administration
- Service Application
- Managed I/O Unit.

There are no Compliance Qualifiers for these categories.

Section A3.7 “Compliance Summary” on page 1194 provides a summary of compliance statements.

A3.2 SERVICE ID

A Service ID is an InfiniBand construct necessary for Communication Management (see volume 1 chapter 12). The Service ID is a component in ComMgt class MADs that the Communication Manager (CM) uses when resolving a connection request or identifying an Unreliable Datagram QP.

This Annex addresses the assignment of Service IDs.

A3.2.1 GOALS AND SCOPE

This Annex defines the structure of the 64-bit IB Service ID name-space (fields and their bit-widths), provides for multiple administration authorities, and clarifies which fields within the Service ID the administration authority controls. It also specifies how various entities may use and interpret the various fields of Service IDs.

This annex specifies structure, semantics, policy, and guidelines governing the use of Service IDs by the various entities involved. It also provides detailed examples of how operating systems and I/O unit firmware need to participate in Service ID administration.

A3.2.2 PRINCIPLES OF SERVICE ID USAGE

Each port on a CA may support a set of services. Each service is identified by a given Service ID. Since not all ports support the same set of services, a tuple composed of the GID, PKey, and a Service ID uniquely identify a service.

In the process of establishing communications, the communication manager behind the port uses the Service ID to identify the service provider. The Service ID in conjunction with a GID identifies a service accessible through that port. In general each Service ID represents a particular service entity and thus identifies the QP consumer providing that service.

Because Service IDs are allocated, advertised, discovered, cached, and eventually used in establishing communication, a number of diverse entities deal with Service IDs in their various roles: server processes, I/O units, I/O controllers, operating systems, service clients (including potential clients), SA, CMs, and boot firmware.

A3.2.2.1 BACKGROUND

A *service* is a function, or set of functions, accessible by a known protocol (i.e., the *service protocol*). An *IB Service* is any service accessible via an IB transport service. A *Service ID* is a value that identifies a particular service. Because a service provider might use multiple QPs, a Service ID identifies the QP consumer (i.e., the application or protocol stack that uses the QPs to provide the service). Thus, it is not the ID of a particular QP. The Service ID provides the means for a service client's communication manager to resolve a given service to a particular QP. The resolution process (how the CM determines which QP) is implementation specific. For example, a service using UD transport service uses a different QP for each partition and a service using the RC transport service needs a different QP for each client. They each can use a single Service ID and the clients use that Service ID to learn which QP to use. In fact, the QP might not exist until the service provider receives a communication request from the client, at which time the CM or service provider creates and configures the QP.

Communication management messages provide the mechanism for CMs to (1) resolve a Service ID to a UD QP, (2) establish a connection to an RC or UC QP for the service identified by a Service ID, and (3) establish a Reliable Datagram Channel to the RD service identified by a Service ID.

The two predominant service models are Client-Server and Peer-Peer.

- For the *Client-Server* model, a service provider (i.e., the application providing the service) offers a service to one or more service clients (i.e., the applications using the service). Typically, service clients make service requests to the service provider and service

providers respond. This implies that the service client knows both the port and the QP where it sends its service requests while the service provider can learn that information from the request.Thus, the service client needs the ability to locate the service provider.

- In the Peer-Peer model, two or more applications (i.e., peers) communicate, and each peer may perform a role similar to a client, a service provider, or both. Peers also need the ability to locate other peers.

For both of these models, an application on one platform wants to locate and establish communications with a particular application on another platform. This is the role of the Service ID. Since there can be multiple instances of a particular service in the IB fabric, it is the combination of the Port GID and Service ID that identifies a particular service provider. Thus, the combination of the port GID, PKey, and Service ID must uniquely identify a single service provider.

The GID used to identify the port providing the service is called the Service GID. A client uses the Service GID to resolve a path to the port. A port can have multiple GIDs and the DGID in the path that the service client uses does not need to be the Service GID. Typically, the service client queries the SA for paths to the service provider's port by specifying the Service GID and uses one of the paths that the SA returns.

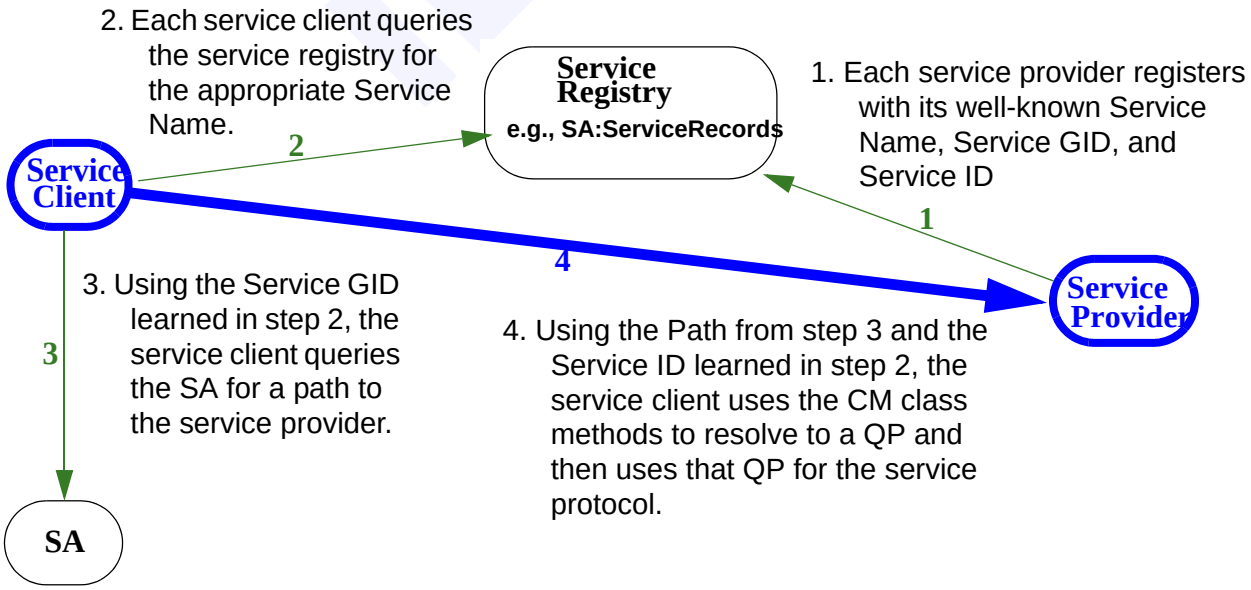


Figure 242 Typical Service Resolution for Host-based Services

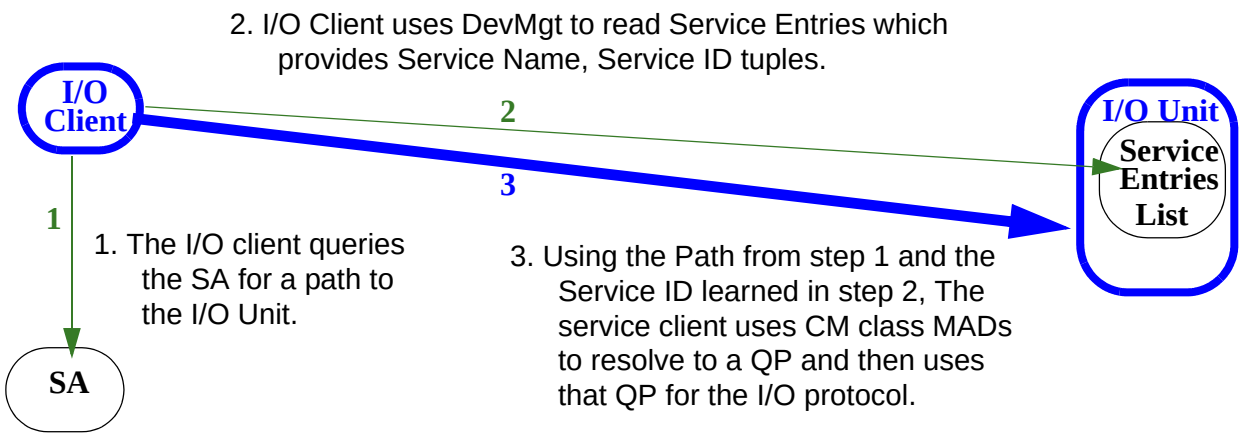


Figure 243 Typical Service Resolution for I/O Services

As previously mentioned, the combination of the {Service GID and Service ID} identifies the port and the service. The SA provides a means for a service provider to register its Service Name, Service GID, Service ID, and other service related information. Service consumers can query the SA for service records for a particular service name and get a list of service providers registered for that Service Name. Each record provides the {Service GID, Service ID} for a service provider. The service consumer uses the Service GID to obtain a path to the service provider (see section 15.2 “SA MADs” subsection on the PathRecord attribute) and then uses the Service ID to resolve the QP (see Chapter 12.6 “Communication Management Messages”).

A3.2.2.2 CONSIDERATIONS

Definition of the Service ID name-space gives special consideration to:

- Scope of the service: Whether a single service instance covers just a port, a channel adapter (or node), multiple channel adapters in a “platform”, etc. -- Well-known (permanent) Service IDs must have a platform wide scope because service clients usually remember these types of services by their host name and Service ID. Dynamically allocated Service IDs can have a port-only scope. Service resolution for services that use dynamic Service IDs typically resolves a Service Name to a <Port GID, Service ID>. While it is possible that a dynamic Service ID has greater than port only scope, the service consumer can not assume a service with a dynamically assigned Service ID is available on any other port. The service provider needs to register its service for each valid {Service GID, Service ID} combination (i.e., the service registers each of its ports).

- Lifetime: How long can a potential client retain a Service ID after discovery and still use it to establish transport service. -- Well-known Service IDs are cacheable and can be remembered forever. However, because of the nature of dynamic Service ID assignment, not all Service IDs have this guarantee. Platforms need to assign Service IDs in a manner that avoids the same Service ID being subsequently assigned to a different service. This annex provides guidance on how to avoid or minimize the possibility of subsequent reassignment (see A3.2.3.3.2 “Coherency Requirements” on page 1184).

- Assigning entity: What body covers the structure of Service IDs for a given service. -- This annex divides the Service ID address space into groups and identifies who controls assignment of those Service IDs. These groups include: local OS, IBTA, and external organizations.

- Accommodating multiple instances behind a port: If multiple instances of a service may overlap in scope or time, how to avoid assigning conflicting Service IDs. -- Where there are multiple entities independently providing different instances of the same service (i.e., same service name/protocol for the same partition), each instance requires a different Service ID value and thus, the application can not use a well-known Service ID.This is because the <platform Name, Well-Known Service ID> or even <Port GID, Well-known Service ID> would not be unique. Where a service is advertised in the form of <Service Name, Service ID>, dynamically assigned Service IDs are sufficient.

- Accommodating host-based services along with device-based services: The two service categories represent two different service models; IPC and I/O. -- IPC and I/O mostly differ on how the client resolves the Service ID (see Figure 242 on page 1176 and Figure 243 on page 1177). Once a client learns the Service ID, there is no difference in how the I/O client verses the IPC client establishes a connection or learns the UD QP (Chapter 12).

- The number of advertisements needed by a single service instance: For services that use a single transport type, it is sufficient to register once for each port through which the service is available. However, an application that supports multiple service types (UC, RC, RD, UD) might need to advertise each service type. It is not necessary for the application to use a different Service ID for each transport service type because in the CM:REQ, the transport service type is specified independently from the Service ID, and the transport service type is implicit in CM:SIDR_REQ MADs. Therefore, the issue becomes one of the service client knowing which transport service type(s) the service provider supports (because Communication can only occur if both ends use the same service type). One solution is to have a dif-

ferent service name for each appropriate transport service type and thus require the service provider to register for each Service Name it supports.

- Using multiple Service IDs to distinguish service types: Even though a protocol could use a different Service ID for each transport service type, the service client still needs to know which transport service type the service provider supports. Different Service IDs typically indicate different service providers.Thus the only reason to use separate Service IDs for the same service is when one QP consumer provides the service for one transport service type and a different QP consumer provides the service for a different transport service type. For all practical purposes they are different services. Each with its own Service ID. A single QP consumer providing the service independent of the selected transport service type constitutes a single service instance and thus only needs one Service ID. The specification defining the service, or defining how the service maps to IB, must specify of how the service consumer knows which service type to use.

A3.2.2.3 ASSIGNING SERVICE IDS

Service IDs fall into different categories on the basis of geographic and temporal scope as shown in Table 343.

Table 343 Service ID Categories and Characteristics

Characteristic	Category: Universally Assigned Well-known Service IDs	Category: Network[a] assigned Service IDs	Category: Locally Assigned Service IDs
Geographical Scope	Global	not specified	platform
Does the Service ID value identify the particular service type?	yes	not specified	no
Temporal Scope: Within the geographical scope, does a Service ID value always indicate the same service type?	yes	not specified	not guaranteed
Can there be multiple independent instances[b] of the same service type per port?	no	not specified	yes

a. Network is loosely defined to mean a subnet or collection of subnets that form a domain where each domain has its own administration responsibilities independent of other domains.
b. Service instance refers to the process providing the service (i.e., the QP consumer) and not individual instances of service objects accessible through the same QP.

Geographical scope refers to whether the Service ID assignment is unique only within the platform, within the network, or universally unique.

- A Service ID which is permanently assigned to a service type (i.e., a particular application, protocol, etc.), to be used everywhere is called a "*well-known*" Service ID and both the service provider and the service client explicitly know which Service ID to use. This annex provides for well-known Service IDs.

  Well-known Service IDs are assigned per service type and all instances of that particular service type, regardless of where they are located, use the same Service ID. Conversely, no other service type uses that Service ID value, so a well-known Service ID value explicitly identifies the service type. Since the {port GID, P_Key, Service ID} tuple must uniquely identify a single service provider, there can be only one service provider for that service type per port per partition.

- Network assigned Service IDs are administered by a network service or network administrator. Network means a subnet or collection of subnets that form a domain where each domain has its own administration responsibilities independent of other domains. IBA does not prescribe any particular network administration model. The Internet Engineering Task Force (IETF) is an organization that defines network service structures and mechanisms. This annex reserves a range of Service IDs for definition by the IETF. The details of how these Service IDs are assigned, as well as their characteristics, are left to the IETF. That definition is not limited to network assigned Service IDs and may include well-known values and locally administered values. This annex also provides for assignment of Service IDs by other organizations.

- *Locally assigned* Service IDs are Service IDs arbitrarily assigned by an agent within the local platform. Locally assigned Service IDs are dynamic. Dynamic means that the Service ID assignment is not guaranteed to last any longer than the immediate lifetime of the service provider or OS (i.e., until the application dies or the OS reboots). Thus, at a later time, the {GID, Service ID} tuple might identify a different service provider. This annex provides for locally assigned Service IDs.

  Each service provider that uses locally assigned Service IDs has a different Service ID value. Thus, there can be multiple independent instances of a particular service type per port. An example of multiple independent instances of a service type is an I/O unit containing multiple I/O controllers of the same type. Note that a single service provider with multiple service objects only needs a single Service ID.

This annex divides the Service ID address space into groups to permit different authorities to independently assign unique Service ID values within their scope.

A3.2.3 SERVICE ID STRUCTURE

The Service ID structure breaks the 64-bit Service ID address space into groups of address spaces, which are administered independently of each other. The first byte of the Service ID identifies the group and thus identifies the administration authority. IBTA defines four of the groups and reserves the remainder for future definition. Figure 244 illustrates the general Service ID structure and Table 343 specifies the values for the Administration Group Number (AGN) field.

First Byte bits 0-7	2nd Byte bits 0-7	3rd Byte bits 0-7	4th Byte bits 0-7	5th Byte bits 0-7	6th Byte bits 0-7	7th Byte bits 0-7	8th Byte bits 0-7
AGN	Dependent on value of AGN field						

Figure 244 General Service ID Structure

Table 344 AGN Codes

Value	Administration Authority
0x00	IBTA Well-known and dynamic Service IDs, see A3.2.3.1 on page 1181
0x01	IETF - (any category), see A3.2.3.2 on page 1183
0x02	Local OS - dynamically assigned with limited lifetime (see A3.2.3.3 on page 1183)
0x10 - 0x1F	External organizations (any category, see A3.2.3.4 on page 1186)
others	reserved

CA3-1: A Service ID assignment shall use the format specified in Figure 244 and use the AGN values specified in Table 344.

A3.2.3.1 IBTA ASSIGNED SERVICE IDS

Certain applications need a preassigned Service ID to avoid requiring each service provider to register or advertise its specific Service ID. The purpose of IBTA Assigned Service IDs is to define Service ID values for IBTA defined protocols. The format of IBTA defined Service IDs is illustrated in Figure 245 and Table 345 lists the IBTA administered Service ID values.

First Byte	2nd Byte	3rd Byte	4th Byte	5th Byte	6th byte	7th byte	8th byte
0x00	Value as per Table 345						

Figure 245 IBTA Assigned Service IDs

Table 345 IBTA Assigned Service IDs

Service	First Byte	2nd Byte	3rd Byte	4th Byte	5th Byte	6th byte	7th Byte	8th Byte
Null	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
ROM Repository	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01
Console Process	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x02
Compliance Testing	0x00	0x00	0x00	0x00	0x00	0x00	0x01	any
SDP Port	0x00	0x00	0x00	0x00	0x00	0x01	Port Number	
	other values reserved							

The ROM Repository, Console Process, and some SDP Port Service IDs are well-known and fully cacheable. Most SDP Port Service IDs are dynamic (refer to the SDP Annex for details). A limitation of Service IDs is that there can be only one service provider using that Service ID per port (or perhaps per platform), and thus, only one instance of an application can use a particular well-known Service ID for that port.

A3.2.3.1.1 NULL SERVICE ID

The null Service ID (all zeros) is a special Service ID that applications may use in data structures containing a Service ID parameter when the Service ID is not relevant. This might be the case when the application does not depend on using CM messages to resolve a UD QP and does not set up a connection. Networking protocols that use multicast to locate service providers is an example where the application does not need a specific Service ID. Use of the null Service ID in CM messages is not valid.

A3.2.3.1.2 ROM REPOSITORY

ROM Repository is a service provided by an I/O unit that allows host nodes to read (and update) device drivers and other images stored in persistent memory on the I/O unit (see Booting Annex). The ROM Repository Service ID identifies the ROM repository application.

A3.2.3.1.3 CONSOLE PROCESS

The IB Console protocol supports both an I/O based and a host based console service (i.e., Console IOC and Console Process). The console IOC does not need or use a well-known Service ID. However, the host based console server process does (see Console Annex). Thus, a console IOC uses locally administrated Service IDs while a host based console server process uses the well-known Console Process Service ID.

A3.2.3.1.4 SERVICE IDS FOR TESTING

IBA reserves a block of 256 Service IDs for compliance testing as specified in Table 345 as “Compliance Testing“. The use of these Service IDs are defined by the IBTA Compliance Test Group.

A3.2.3.1.5 SOCKETS DIRECT PROTOCOL

Sockets Direct Protocol (SDP) is a network protocol equivalent to TCP that uses a Reliable Connection QP for each socket. A socket is identified by its TCP port number. The set of SDP Port Service IDs identify specific applications by their equivalent TCP/IP port number. The last two bytes of the SDP Port Service ID is the equivalent TCP port number.

Some SDP Port values are well-known port numbers and fall into the category of well-known Service IDs as per Table 343 on page 1179. Other values have the characteristic of a dynamically assigned Service ID (i.e., fall into locally assigned category of Service IDs as per Table 343 on page 1179). Refer to the Sockets Direct Protocol Annex for details.

A3.2.3.2 IETF SERVICE IDS

The IBTA recognizes the importance of the IETF and reserves a range of Service IDs for that organization. How the IETF chooses to use this address space is left entirely to the IETF. Unless the IETF specifies otherwise, these Service IDs should not be considered cacheable.

First Byte	2nd Byte	3rd Byte	4th Byte	5th Byte	6th byte	7th byte	8th byte
0x01	any value						

Figure 246 IETF Service IDs

A3.2.3.3 LOCAL OS ADMINISTERED SERVICE IDS

Local OS administered Service IDs are those Service ID values assigned by the local platform. Each platform assigns locally administered Service IDs independent of other platforms. In a host, this assignment is the responsibility of the operating system (or its designated agent, such as the CM) that resides above the verbs layer. An equivalent means of assigning Service IDs can exist for an I/O unit. Other than the AGN field, IBA does not impose any particular format on local OS administered Service IDs, leaving that task to the discretion of the local OS. Figure 247 illustrates the structure of local OS administered Service IDs.

First Byte	2nd Byte	3rd Byte	4th Byte	5th Byte	6th byte	7th byte	8th byte
0x02	Dynamic value assigned by local OS						

Figure 247 OS Administered Service IDs

Locally administered Service IDs have local scope. That means that the same service type can have a different Service ID value on each platform that provides that service. Thus, a service client needs a way to learn the Service ID. Learning the Service ID can be handled though a number of service registration facilities such as a DHCP[28] server, a registration service such as the SubnAdm Service Records, and by Device Management class Service Entries (see A3.2.4 “Resolving Service Names” on page 1187).

A3.2.3.3.1 PORT ASSOCIATION

Dynamically assigned Service ID assignment is per port. Whether the OS assigns a dynamic Service ID to an application on a per port basis or per platform basis is a matter of OS policy. This means that an OS could assign a Service ID to an application and restrict that application to a particular port (or set of ports) and assign the same Service ID to other applications as long as none of the port assignments overlap.

CA3-2: The assignment of Service IDs shall ensure that the <Port GID, P_Key, Service ID> tuple is unique and unambiguous. That is, there is at most one application (or a set of applications that know that they are sharing the Service ID) that believes an incoming CM MAD on that port containing that Service ID is a request for its service.

CA3-3: A CM shall not redirect a CM:REQ or CM:SIDR_REQ to another port unless the <Port GID, Service ID> tuple for both ports reference the same QP consumer on the responder side.

A3.2.3.3.2 COHERENCY REQUIREMENTS

A primary concern with using Service IDs is their relative lifetime. That is, once a service client learns a Service ID, for what period of time is the Service ID valid? How the platform administers Service IDs affects the answer.

A cacheable Service ID is a Service ID value where the {ServiceGID, Service ID} tuple always represents the same service entity. Only well-known Service IDs are cacheable. Dynamic Service IDs do not have the same coherency guarantees.

Because locally administered Service IDs are dynamically assigned, there is no guarantee that an application will be assigned the same Service ID each time the platform, OS, or process initializes. This means that a particular {Service GID, Service ID} tuple could subsequently identify a different application. A coherency problem could exist if a Service ID value assigned to one application is subsequently assigned to another applica-

28. Dynamic Host Configuration Protocol

tion while service clients of the first application still think that the Service ID identifies the first application.

The assigning authority has the responsibility of not reassigning a Service ID value until all clients have stopped using it. Just because an application terminates, does not mean that its Service IDs are not in use. In fact, there may be service registries that retain Service IDs for long periods of time, depending on lease periods established for those service listings. These lease periods could be any period of time (e.g., minutes, days, etc.), and not all service registries have leases.

Service records registered with the SA contain a lease period that identifies how long that service record is valid. The expectation is that the service continually renews that lease while the application is active and the Service ID is valid. If for any reason the Service ID assignment changes (e.g. the platform providing the service resets, the application terminates, etc.), the lease expires and the SA removes the service record from use.

Some steps that a platform can take to avoid a coherency problem are:

- Persistently assign Service IDs so that an application always uses the same value.
- Assign Service IDs sequentially, storing the next value to be assigned in non-volatile storage, so it survives power cycles and platform resets. Thus each time the platform resets, applications are assigned a new set of Service IDs that do not overlap with the previous instantiation. Due to the large local OS Service ID address space (2^{56} values), the OS should never have to reuse a value.
- Assign Service IDs sequentially, starting with a random number each time the OS starts up (or resets). The probability of reassigning a Service ID is extremely small and the probability of a coherency problem is even smaller - much better than exists in networking such as for TCP and UDP port numbers.
- In all cases, from the time the platform initializes until it resets, it should never reassign a Service ID value to a different application, unless the original application has specifically indicated that it is ok for the OS to reuse the value.
- A platform that reassigns a Service ID value should assure that there are no SA:ServiceRecords for the old assignment. One way this can be meet is by restricting the maximum SA:ServiceRecord lease period when creating/modifying a ServiceRecord and not reassigning the Service ID until after the lease has expired.

When the client receives a ServiceRecord from the SA, the ServiceRecord contains a ServiceLease component that indicates the remaining

amount of time the Service ID will remain valid. A client should not use a Service ID after its lease period expires.

A3.2.3.4 EXTERNALLY ADMINISTRATED SERVICE IDS

Other organizations may wish to assign Service IDs or need to map their equivalent of a Service ID to an IBA Service ID. To keep the IBTA out of the business of administering Service IDs to other organizations, any organization that has an IEEE Organizational Unique Identifier (OUI)[29] can specify its own Service IDs as long as they follow the format specified in Figure 248.

First Byte		2nd Byte	3rd Byte	4th Byte	5th Byte	6th byte	7th Byte	8th Byte
bits 4-7	bits 0-3	bits 0-7	bits 0-7	bits 0-7	bits 0-7	bits 0-7	bits 0-7	bits 0-7
0x1	org[a]	OUI			Value assigned by organization[a].			

Figure 248 Externally Administrated Service IDs

a. value assigned by the organization indicated by the OUI field

OUI = IEEE 24-bit Organization Unique Identifier

By placing the organization's IEEE assigned 24-bit Organization Unique Identifier in bytes 2, 3 and 4, the organization can define Service IDs by specifying the value for bits 0-3 of the first byte and all bits of the last 4 bytes. For example, the OUI assigned to ANSI X3 is 00609E and thus if any X3 subgroups, such as X3T10, needs its own Service IDs, X3 can create them by using its OUI. Thus, Service IDs 0x1h00609Ehhhhhhhh (where h is any hexadecimal digit) can be assigned by X3.

Since these values are application specific, the application knows if the Service ID is cacheable or not. Unless explicitly known, these Service IDs should not be considered cacheable.

CA3-4: Other than the IETF, which may specify Service IDs as per Section A3.2.3.2, an external organization that specifies Service IDs shall use the structure specified in Figure 248.

CA3-5: The OUI in an Externally Administrated Service ID shall be an OUI assigned to the defining organization by the IEEE or an OUI which that organization has permission to use.

29. For additional details, see: "IEEE OUI and Company_id Assignments" at www.standards.ieee.org/regauth/oui/

A3.2.4 RESOLVING SERVICE NAMES

Service resolution is the process of learning the *service location* (i.e., the port GID and Service ID) for a particular service instance.

A3.2.4.1 SERVICE ADVERTISEMENT

One way to resolve a service location is through service advertisement. A *service advertisement* is where the service provider places service information in a known location so service clients or peers can find it. The service information includes <service name, port GID, and Service ID>. Services are advertised in different ways.

- A managed I/O unit advertises I/O services provided by its I/O controllers via DevMgt ServiceEntry or ServiceRecords attributes. These records list <service name, Service ID> tuples for the I/O controller. The port GID is implied as the same port that the client used to read the DevMgt attribute.
- Host-based IB subnet services are advertised in SA ServiceRecords. These records list <service name, port GID, Service ID> tuples.
- Some network based services are advertised in configuration and directory servers, such as a DHCP server / Directory Agent where service clients retrieve service records through standard service location protocols (e.g., SLP v2).

Each service advertisement describes exactly one service instance.

Once the service's port GID and Service ID are known, the service client can:

- Resolve the UD QP & Q_Key by sending a CM:SIDR_REQ to that port specifying the Service ID.
- Resolve the RD QP/RDC & Q_Key by sending a CM:REQ to that port specifying the Service ID.
- Create an RC or UC connection by sending a CM:REQ to that port specifying the Service ID.

A3.2.4.2 MULTICAST QUERY

Another means to resolve a service location is via multicast. Naturally this only works for services using Unreliable Datagram or Raw Datagram transport service. The first service provider creates a multicast group and additional service providers join the MC group. The service client then sends a multicast datagram to that MC group address identifying the particular service provider (e.g., service name, host name, protocol ID, etc.), and the appropriate service provider(s) respond to it. The service provider learns the location of the service client from the multicast datagram and responds with a unicast datagram. The service client learns the location

of the service provider from the response. For the multicast method, a Service ID is not necessary.

The multicast method implies that the service providers and service clients know which multicast address to use. There are several possible ways to solve this problem. One way is to use service advertisement described previously.

For example, if the IETF wanted to define an IP over InfiniBand (IPoIB) service resolution process, it could define a unique service name such as "xyz.MCGroup.IETF"[30] and require that the first service provider (or router) create a MC group and register it via a SubnAdmSet(ServiceRecord) using ServiceName= "xyz.MCGroup.IETF", ServiceGID = the assigned multicast GID (in text notation), and Service ID = null or well-known Service ID.

Now all interested parties can learn the multicast address by:

a) Querying the SA for ServiceRecords with ServiceName = "xyz.MCGroup.IETF"

b) Querying the SA for MCGroupRecords with MGID=ServiceGID (after converting the ServiceGID string to its 128-bit binary equivalent).

The MCGroupRecord provides all of the parameters the client needs to send a UD multicast packet.

A3.2.4.3 ALTERNATIVES

Not every service type needs to use advertisement or multicast. Certain service types are *ubiquitous* (i.e., present on most ports or present on platforms of a certain type: *e.g.* the service for accessing the ROM Repository is found on some managed I/O units, an OS-specific service on all hosts incorporating a particular OS, etc. For these situations, there are various ways for the client to learn the GID of service provider and it is common for the service to employ a *well-known Service ID*.

The method that a service client uses to learn which platform provides the service is application dependant. However, it is typical for service clients using well-known Service IDs to be configured with just the name of the host providing the service, and then query a service that resolves the host name to a port GID.

30. This is a fictitious name created for the example and does not imply an actual service name.

A3.3 I/O CONTROLLER IDENTIFICATION

The DevMgt class attributes contain components that identify properties of an I/O controller.These components are: VendorID, DeviceID, Device Version, Subsystem VendorID, SubsystemID, IO Class, IO Subclass, Protocol, Protocol Version, ID String.

The I/O controller vendor provides this information so the host can identify the I/O controller and match it with an appropriate I/O driver (refer to the I/O annex). This section describes the acceptable practice for deriving values for these fields.

A3.3.1 VENDOR INFORMATION

The following components are vendor specific: *VendorID, DeviceID, Device Version, Subsystem VendorID, SubsystemID, ID String*.

The vendor places its IEEE assigned Organization Unique Identifier (OUI)[31] in the *VendorId* field and may place any value in the *DeviceID* and *Device Version* fields. The vendor may also provide an ASCII string of its choice in the *ID String* field.

The *Subsystem VendorID* and *SubsystemID* provide additional information when a subsystem vendor uses components provided by other vendors. In this case the subsystem vendor provides its OUI in the *Subsystem VendorID* field and may specify any value in the *SubsystemD* field.

A vendor that produces a generic controller (i.e., one that supports a standard I/O protocol such as SRP), which does not have vendor specific device drivers, may use the value of 0xFFFFFF in the *VendorID* field. However, such a value prevents the vendor from ever providing vendor specific drivers for the product.

A3.3.2 GENERIC INFORMATION

Generic information fields refer to the *IO Class, IO Subclass, Protocol*, and *Protocol Version* components of the in DevMgt attributes. An I/O Controller (IOC) uses these fields to indicate that it supports a standard I/O protocol[32]. A host uses these fields to match the IOC with an I/O driver that performs that I/O protocol (see Annex A1: “I/O Infrastructure” on page 1121 for driver matching).

31. For additional details, see: “IEEE OUI and Company_id Assignments” at www.standards.ieee.org/regauth/oui/

32. A standard I/O protocol refers to a protocol definition that permits a vendor other than the IOC vendor to provide the I/O driver. Thus, vendor and product specific information is not pertinent in matching the IOC with an I/O driver.

CA3-6: A managed I/O unit shall implement a Device Management Agent as per Section 16.3 (or per a supplemental Annex that supersedes Section 16.3).

CA3-7: An IOC shall specify Class, Subclass, and Protocol values in accordance with A3.3 I/O Controller Identification.

It is the combination of the Class, Subclass, and Protocol fields that identifies a single I/O class protocol.

The most significant byte of the Class field identifies the I/O category and the least significant byte of the Class field and the SubClass field identify the defining organization as illustrated in Figure 249.

Field:	**Class Field**			**SubClass Field**
Bits:	15-12	11-8	7-0	15-0
Content:	Category Table 346	reserved	OrganizationID[a]	

Figure 249 Class/Subclass fields for External Protocols

a. The most significant byte of the OUI is in the least significant byte of the Class field and the least significant byte of the OUI is in the least significant byte of Subclass field. Bits are in canonical order.

Table 346 specifies the values for Category:

Table 346 I/O Category

Value	I/O Category
0x0	none or other
0x1	Storage class
0x2	Network class
0x4	Video/Multimedia class
0xF	Unknown or multiple classes
others	reserved

A3.3.3 IBTA PROTOCOLS

An OrganizationID code of 0xFFFFFF indicates an IBA defined class protocol as listed in Table 347.

Table 347 IBTA Defined Protocols

Class[a]	Subclass	Protocol[b]	Definition
0xzzFF	0xFFFF	0xFFFF	The IOC does not support a standard I/O protocol (only a proprietary protocol)
0x40FF	0xFFFF	0x0001	Console protocol as defined in Console Annex

a. z = any value
b. All other combinations reserved

A3.3.4 OTHER PROTOCOLS

Any organization may define their own protocol identifiers by specifying a Class, SubClass, Protocol tuple using their OrganizationID in the Class field (i.e., the least significant 8 bits) and the Subclass field, as illustrated in Figure 249. The OrganizationID is the IEEE Organization Unique Identifier[33] (OUI) assigned to the organization that is defining the Protocol and Version field values. Thus any organization that has an IEEE assigned OUI may specify Protocol codes. For each Protocol value, the defining organization selects a Category. It also maintains control of defining Version values.

A3.4 SERVICE NAMES

Service names appear in a number of different places. In particular, in SubnAdm:ServiceRecord attributes and in DevMgt:ServiceEntries attributes. The recommended practice is that the service name end in a name unique to the assigning authority.

The InfiniBand Trade Association (IBTA) reserves the value ".IBTA" for IBTA defined service names. Thus, the IBTA is the only organization that may use that value in a service name. For example, "Console.IBTA" can only be assigned by the IBTA.

CA3-8: The value ".IBTA" shall not be used in a service name unless the name is explicitly defined by the InfiniBand Trade Association.

CA3-9: An application that specifies an IBTA defined protocol name shall conform to the requirements for the associated protocol.

33. For additional details, see: "IEEE OUI and Company_id Assignments" at www.standards.ieee.org/regauth/oui/

A3.4.1 IBTA SERVICE NAMES

Table 348 lists IBTA Service Names.

Table 348 IBTA Service Names

Service Name	Defined in:
BaseboardManager.IBTA	Chapter 16 - Baseboard Management
BIS.IBTA	Boot Information Service Annex
BootManager.IBTA	Booting Annex
Console.IBTA	Annex A2 “Console Service Protocol”
DeviceManager.IBTA	Chapter 16 - Device Management

A3.4.2 I/O SERVICE RECORDS

Each IOC provides a list of protocol identifiers in one or more service entry attributes. The number of attribute records for a particular IOC is indicated in the IOControllerProfile. Typically, an I/O driver uses this information for establishing connections with the IOC.

For each protocol the IOC supports, the IOC provides a service entry. Each entry contains a ServiceName and its associated ServiceID. Service names need to be unique within the scope of the IOC (i.e., the IOC can not list the same service name more than once in any of its service entries). For proprietary drivers, the IOC vendor defines a service name and for standard I/O protocols, the organization defining the I/O protocol defines the service name.

A3.4.3 SERVICERECORD ATTRIBUTE

The SubnAdm:ServiceRecord attribute provides the means for a service to advertise its presence. The service record contains a Service Name and its associated Service GID and Service ID. Typically a service client queries the SA for ServiceRecords matching a particular service name to locate a port and Service ID for a particular protocol. The organization defining the protocol defines the Service Name.

A3.5 MANAGEMENT CLASS CODES

Chapter 13 specifies a range of management class codes as application specific. Table 349 lists application specific management class code values and identifies their respective management class.

Table 349 Application Specific Management Class Codes

Class Code[a]	Management Class
0x10	Device Administration - see Configuration Management Annex
0x11	Boot Management - see Booting Annex
0x12	Boot Information Service - see Boot Information Service Annex
0x20	Reserved for Compliance and Inter-operability Testing
0x21	Congestion Management - see Congestion Management Annex

a. All other values reserved for future assignment

A3.6 QUEUE KEYS

Chapter 10 section 10.2.5 specifies that Q_Keys with the most significant bit set can only be sent by QPs that have that Q_Key in the QP context. This provides for a set of 'privileged Q_Keys'. That is, the only way for a QP to send or receive a privileged Q_Key is to have been created with that privileged Q_Key. Since the operating system controls QP creation and the user level application can not alter the QP context, the OS has the means to control use of privileged Q_Keys.

In order for privileged Q_Keys to be meaningful, their use needs to be restricted to the applications for which the Q_Key is assigned. Table 350 lists Privileged Q_Keys and their assigned use.

Table 350 Privileged Q_Keys

Q_Key	Management Class
0x8000_0000 thru 0x8000_FFFF	Available for general use - can be used without permission from the InfiniBand Trade Association
0x8001_0000	Well known IB management Q_Key
0x8001_0001	Device Administration Agent (see Configuration Management Annex)
0x8001_0000 thru 0x8001_FFFF	reserved for IBTA definition
0x8002_0000 thru 0x8FFF_FFFF	Reserved for external use - An organization can request a privileged Q_Key by sending the request to the InfiniBand Trade Association - attention Application Working Group.

A3.7 COMPLIANCE SUMMARY

This annex specifies 3 new Compliance Categories (see Volume 1 Chapter 20 for explanation of compliance categories and qualifiers).

These new categories are:

- Service ID Administration
- Service Application
- Managed I/O Unit.

There are no Compliance Qualifiers for these categories.

A3.7.1 SERVICE ID ADMINISTRATION

In order to claim compliance to the InfiniBand Architecture Specification for the Compliance Category of Service ID Administration, a product shall meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support (currently there are no optional compliance qualifiers for this category).

A3.7.2 SERVICE APPLICATION

In order to claim compliance to the InfiniBand Architecture Specification for the Compliance Category of Service Application, a product shall meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support (currently there are no optional compliance qualifiers for this category).

A3.7.3 MANAGED I/O UNIT

In order to claim compliance to the InfiniBand Architecture Specification for the Compliance Category of Managed I/O Unit, a product shall meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support (currently there are no optional compliance qualifiers for this category).

ANNEX A4: SOCKETS DIRECT PROTOCOL (SDP)

A4.1 INTRODUCTION

This document is a supplement to release 1.0.a of Volume 1 of the InfiniBand Architecture, herein referred to as the base document. This document provides an annex to the base document defining a transport layer protocol called Sockets Direct Protocol (SDP).

SDP is a byte-stream transport protocol that closely mimics TCP's stream semantics. SDP utilizes InfiniBand's advanced protocol offload, kernel bypass, and zero copy capabilities. Because of this, SDP can have lower CPU and memory bandwidth utilization when compared to conventional implementations of TCP, while preserving the familiar byte-stream oriented semantics upon which most current network applications depend.

This version of the specification expands the SDP protocol to leverage the optional *Base Memory Management Extensions* provided in this version of the IBA Specification.

Base Memory Management Extensions provided in this version of the IBA Specification support *Local and Remote Invalidate* features. Connecting and accepting peers in a SDP connection will need to exchange the *Send with Invalidate* capability during the connection setup. Once both sides of an SDP connection have advertised *Send with Invalidate* support, both sides shall use R_Keys that are suitable for a *Remote Invalidate.*

This version of the specification allows each peer to support more than 255 outstanding Zcopy advertisements.

A4.1.1 ARCHITECTURAL GOALS

SDP has the following architectural goals:

- Maintain traditional sockets SOCK_STREAM semantics as commonly implemented over TCP/IP. Some specific issues that are addressed include:
  - Graceful close, including half-closed sockets
  - Ability to use TCP port space
  - IP addressing (IPv4 or IPv6)
  - Connecting/accepting connect model
  - Out-of-band (OOB) data
  - Support for common socket options

- Support for byte-streaming over a message passing protocol
  - Capable of supporting kernel bypass data transfers
  - Capable of supporting zero-copy data transfers from send upper-layer-protocol (ULP) buffers to receive ULP buffers.

This specification focuses specifically on the wire protocol, finite state machine, and packet semantics. Operating system specific issues and other implementation specific issues are outside the scope of this specification, including application programming interfaces (APIs), ULP completion mechanisms, kernel bypass capabilities, etc. These issues are left up to each implementation.

Note that SDP only supports SOCK_STREAM semantics (i.e., byte-stream), not SOCK_DGRAM (i.e., datagram) semantics.

A4.1.2 OVERVIEW OF THE BYTE-STREAM PROTOCOL

SDP's ULP interface is a byte-stream interface that is layered on top of InfiniBand's Reliable Connection message-oriented transfer model. The mapping of the byte-stream protocol to InfiniBand message-oriented semantics was designed to enable ULP data to be transferred by one of two methods - through intermediate private buffers (Bcopy) or directly between ULP buffers (Zcopy).

A mix of InfiniBand Send and RDMA mechanisms are used to transfer ULP data. Zcopy uses InfiniBand RDMA Reads or Writes, transferring data between RDMA buffers. Bcopy uses InfiniBand Sends, transferring data between send buffers and receive private buffers. An implementation is expected, but not required, to have RDMA buffers be ULP buffers, enabling the RDMA path to perform a true zero-copy. An implementation is expected, but not required, to use the receive private buffer pool to buffer data and eventually copy the data from the receive private buffer pool into the receive ULP buffer. An implementation may choose to implement different ULP buffering semantics.

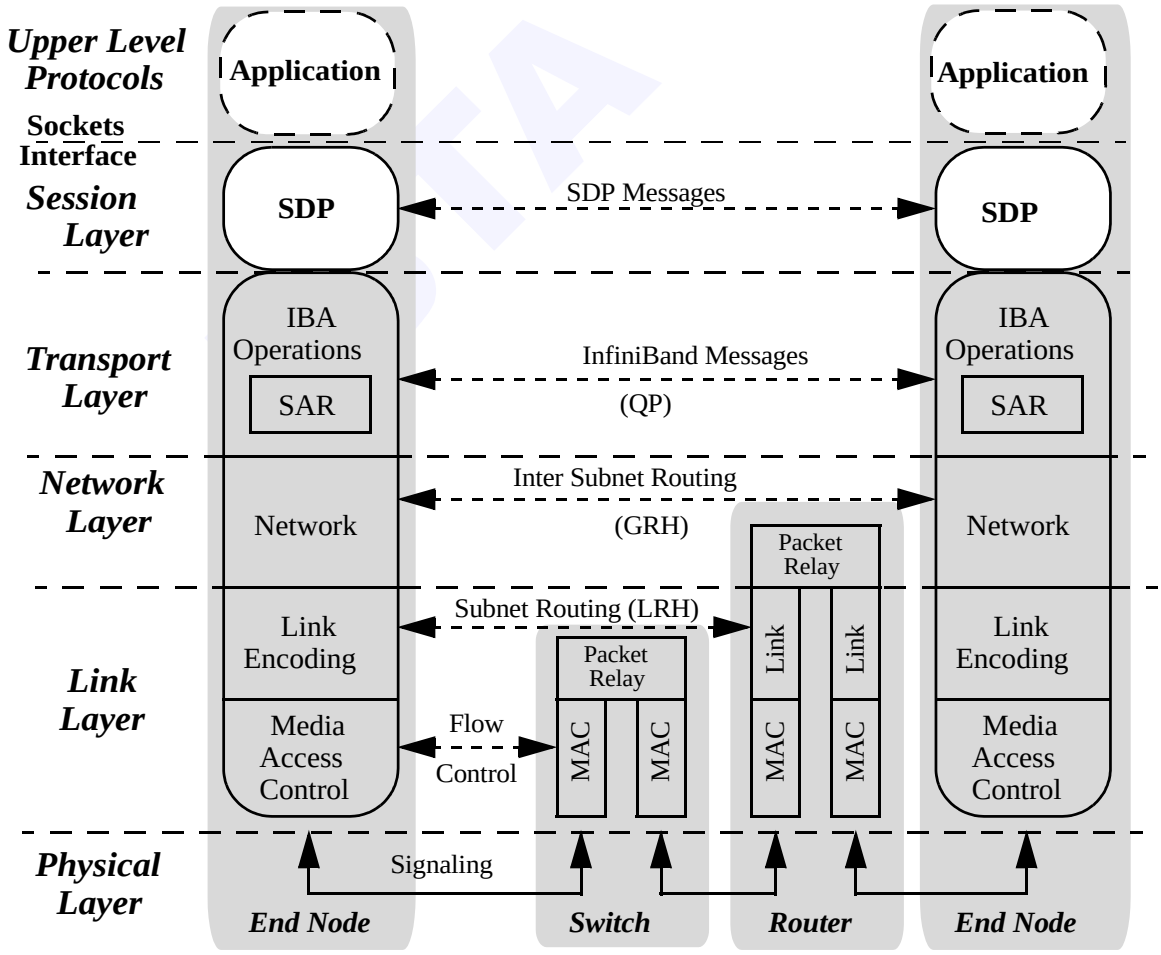
SDP has two types of buffers:

- Private buffers - used for transmission of all SDP messages and ULP data that is to be copied into the receive ULP buffer. The Bcopy data transfer mechanism is used for this traffic.
- RDMA buffers - used when performing Zcopy data transfer. ULP data is intended to be RDMAed directly from the Data Source's ULP buffer to the Data Sink's ULP buffer.

The policy which controls when to use private buffers versus RDMA buffers is outside the scope of this specification. An implementation dependent parameter defined as the Bcopy Threshold is used to abstractly define the results of the policy decision.

To remain compatible with the base InfiniBand specification, this annex defines a wire protocol which can be layered on top of the semantics defined by verbs (see section 11.1 Verbs Introduction and Overview on page 546 in Volume 1a and Figure 250 SDP Relation to IBA Architecture Layers on page 1197). Thus when this specification uses the term “send” in relation to an SDP message, it is in reference to when the Work Queue Entry is posted to the Send Queue. When this specification uses the term “receive” in relation to an SDP message, it is in reference to when the associated Completion Queue Entry is dequeued. Sending an SDP message specifically does not refer to when the CA actually places the message on the InfiniBand link; reception of an SDP message specifically does not refer to reception of the SDP message at the CA interface or when the SDP message Completion Queue Entry is first inserted into the Completion Queue.

Figure 250 SDP Relation to IBA Architecture Layers



A4.2 GLOSSARY

Accepting peer
The peer which sent the REP MAD during connection establishment. Equivalent to the passive peer in section 12.9.6 Communication Establishment - Passive on page 688.

Bcopy
See Buffer-copy.

Buffered Mode
One of three modes for an SDP half-connection. This mode uses the Bcopy Data transfer mechanism exclusively.

Bcopy Threshold
A locally defined threshold existing separately for each peer of a half-connection, which helps the peer determine whether it will attempt to use the Bcopy Data transfer mechanism versus the Zcopy Data transfer mechanism to transfer a given size ULP buffer.

Buffer-copy
A Data transfer mechanism where the transfer of ULP payload between peers is done through an SDP managed receive Private Buffer pool. The received ULP data may require a copy into the receive ULP Buffer.

Combined Mode
One of three modes for an SDP half-connection. This mode enables the use of the Bcopy and Read Zcopy Data transfer mechanisms.

Connecting peer
The peer which received the REP MAD during connection establishment. Equivalent to the active peer in section 12.9.5 Communication Establishment and Release - Active on page 687.

Controlling address space
The address space in which the socket currently exists. This is relevant for socket duplication, where a Non-controlling address space can request control of the socket.

Data Sink
The peer receiving ULP payload. Note that the Data Sink can be required to both send and receive messages to complete a data transfer mechanism.

Data Source
The peer sending ULP payload. Note that the Data Source can be required to both send and receive messages to complete a data transfer mechanism.

Data transfer mechanism
A sequence of messages, including control messages (with or without ULP payload) and/or RDMA messages, to transfer data from a Data Source to a Data Sink with flow control. Four data transfer mechanisms are defined - Buffer-copy, Zero-copy with RDMA Write (Write Zcopy), Zero-copy with RDMA Read (Read Zcopy), and Transaction.

Flow control mode
The mode of the half-connection that determines which Data transfer mechanisms may be used; can be either Buffered, Pipelined, or Combined.

In-Process — An SDP message sequence is In-Process if it is actively being worked on. For example, if a SrcAvail message is In-Process, RDMA reads may have been issued or completed, but a completion message (RdmaRdCompl or SendSm) has not been sent. See also Unprocessed and Processed.

Incomplete — See In-Process.

Message — In this annex "message" is context dependent. It may either be an SDP message or an InfiniBand message. If there is any ambiguity, either the exact message name is used or it is explicitly stated whether it is an SDP or an InfiniBand message.

Mode — See Flow control mode.

Mode Master — The side of an SDP half-connection which may initiate a mode transition by sending a ModeChange message. This is either the Data Source or the Data Sink, depending upon the flow control mode.

Mode Slave — The side of an SDP half-connection which reacts to mode changes. It may advise the Mode Master to change modes (by using SendSm messages or setting the REQ_PIPE flag in the BSDH), but it cannot force a mode change. This is either the Data Source or the Data Sink, depending upon the flow control mode.

Non-controlling address space — An address space that does not currently have control of the socket. See Controlling address space.

OOB — See Out-Of-Band Data.

Out-Of-Band Data — Out-of-Band Data is a single byte of data in the data stream whose handling should be expedited.

Pipelined Mode — One of three modes for an SDP half-connection.This mode enables the use of the Bcopy, Read Zcopy, Write Zcopy, and Transaction Data transfer mechanisms.

Private Buffer — Buffers owned by SDP and not exposed to the ULP. Receive private buffers are used for reception of all SDP messages and ULP data transfer using the Bcopy or Transaction Data transfer mechanisms. All receive private buffers must be at least the current advertised size, and are posted to the InfiniBand Receive Queue.

Processed — An SDP message sequence has been completed by sending the last message of the sequence. Note that the completing message may not have been received. For example, a SinkAvail advertisement is said to have been Processed when the corresponding RdmaWrCompl message has been sent. See also Unprocessed and In-Process.

RDMA Buffer — A buffer which is exposed by the SDP protocol for RDMA access. It is used by the Write Zcopy, Read Zcopy, and Transaction Data transfer mechanisms.

Receiver — Destination of an SDP or InfiniBand message.

SDP Message — An InfiniBand message which contains an SDP Base Sockets Direct Header (BSDH). This specifically does not include InfiniBand RDMA Write and RDMA Read messages. SDP connection initialization messages are encapsulated in the CM REQ and REP messages.

Sender — Source of an SDP or InfiniBand message.

Transaction — A Data transfer mechanism that collapses a Data message and a SinkAvail message into a single SDP message.

ULP — Upper Layer Protocol.

ULP Buffer — Buffers owned by and visible to the ULP. A ULP buffer may serve as an RDMA source buffer, an RDMA sink buffer, or a send buffer.

Unprocessed — An SDP message is Unprocessed if it has been sent, and possibly received, but it has not been operated on. For example, a SinkAvail message is Unprocessed if it has been sent by the Data Sink, received by the Data Source, but no RDMA Writes have begun. Note that processing of flow control information by the receiver may have been done. See also In-Process and Processed.

WrapSubtract — WrapSubtract is meant to represent a function which subtracts the second argument (arg2) from the first argument (arg1). Both arguments are unsigned integers which wrap from a value of 0xFFFFFFF to 0x0. Mathematical operations on wrapping unsigned integers can be done by a variety of methods, including methods defined in RFC1982 (see www.ietf.org). The following equation is an example implementation of the function that casts the unsigned integers into two's complement integers, and then takes the absolute value of the result:

```
result = abs((int)arg1 - (int)arg2)
```

Zcopy — See Zero-copy.

Zero-copy — Three Data transfer mechanisms (Read Zcopy, Write Zcopy, and Transactions) where the transfer of ULP payload between peers is done directly into the ULP buffer, thus avoiding a Buffer-copy on receive.

A4.3 SDP MESSAGE FORMATS

Sockets Direct Protocol defines several types of messages to transfer data and control the state of a connection. Each SDP message contains a Base Sockets Direct Header (BSDH). SDP connection setup messages (Hello and HelloAck) are encapsulated within CM REQ and REP MADs, respectively. Some SDP message types also contain an extended header and/or ULP payload. The extended header (if present) immediately follows the BSDH. The ULP payload follows the headers (BSDH and extended header, if any).

CA4-1: All SDP message headers **shall** use big endian byte ordering, as defined in section 1.5.1 Byte Ordering on page 66.

CA4-2: The SDP Hello and HelloAck messages **shall** be carried in the private data of the CM REQ and CM REP MADs, respectively.

CA4-3: All SDP messages, except for Hello and HelloAck, **shall** be transmitted via the InfiniBand RC channel.

A4.3.1 BASE SOCKETS DIRECT HEADER (BSDH)

CA4-4: All SDP messages **shall** contain the Base Sockets Direct Header, as defined in this section.

Figure 251 Base Sockets Direct Header (BSDH)

bits / bytes	31-24	23-16	15-8	7-0
0-3	MID	flags	bufs	
4-7	len			
8-11	MSeq			
12-15	MSeqAck			

A4.3.1.1 MESSAGE IDENTIFIER (MID)

Specifies the type of the SDP message. The type of SDP message indicates whether an extension header is present.

CA4-5: Table 351 **shall** be used to define the MID parameter in the BSDH, the type of SDP message, and the extended headers and payload that follow the BSDH for a specific SDP message.

Table 351 SDP Message Definitions

MID[7-0]	Message Name	Extended Header Following the Base Sockets Direct Header	Packet Contents Following the Extended Header (if any)
00000000	Hello	HH	none
00000001	HelloAck	HAH	none
00000010	DisConn	none	none
00000011	AbortConn	none	none
00000100	SendSm	none	none
00000101	RdmaWrCompl	RWCH	none
00000110	RdmaRdCompl	RRCH	none
00000111	ModeChange	MCH	none
00001000	SrcAvailCancel	none	none
00001001	SinkAvailCancel	none	none
00001010	SinkCancelAck	none	none
00001011	ChRcvBuf	CRBH	none
00001100	ChRcvBufAck	CRBAH	none
00001101	SuspComm	SuspCH	none
00001110	SuspCommAck	none	none
00001111	Reserved	n/a	n/a
00010000-00111111	Reserved	n/a	n/a
01000000-01111111	Experimental	n/a	optional payload
10000000-11111100	Reserved	n/a	n/a
11111101	SinkAvail	SinkAH	optional ULP payload
11111110	SrcAvail	SrcAH	optional ULP payload
11111111	Data		optional ULP payload

Note that SDP messages Hello and HelloAck are encapsulated in the private data of CM REQ and REP MADs, respectively.

Reserved MID values may be assigned in future versions of the protocol. Experimental values will never be used for permanent assignment.

A4.3.1.2 FLAGS

CA4-6: Table 352 **shall** be used to define the BSDH Flags field.

Table 352 BSDH Flags

Bit Position	Name	Description
0	OOB_PRES	Out-Of-Band Data is present
1	OOB_PEND	Out-Of-Band Data is pending
2	REQ_PIPE	Request change to Pipelined Mode
3-7	reserved	transmitted as zero and not checked at receiver

If the OOB_PRES bit is set then the last byte of the ULP payload in the SDP message is OOB data.

CA4-7: The OOB_PRES bit **shall** be set only in a Data message.

Normal ULP payload may also be present in the Data message before the OOB data. See section A4.6.5.3 Processing Out-Of-Band Data on page 1240.

If the OOB_PEND bit is set then Out-Of-Band data has been sent by the ULP. This flag may be set in any SDP message. The actual Out-Of-Band data may or may not be in the current SDP message. See section A4.6.5.3 Processing Out-Of-Band Data on page 1240.

The REQ_PIPE bit is a hint from the Data Sink to the Data Source to switch to Pipelined Mode.

CA4-8: The Data Sink **shall** clear the REQ_PIPE bit if it would prefer the Data Source to stay in Combined Mode. The Data Sink **shall** set the REQ_PIPE bit to one if it would prefer the Data Source to switch to Pipelined Mode or remain in Pipelined Mode. REQ_PIPE **shall** only be set in RdmaRdCompl messages.

The Data Source is not obligated to follow the recommendation.

CA4-9: The Data Source **shall** ignore REQ_PIPE if the current mode is Buffered Mode.

A4.3.1.3 BUFFERS (BUFS)

Number of private buffers which were currently posted after the last SDP message was received by the local peer, in units of private buffers. More precisely, Bufs equals the total number of private buffers posted over the lifetime of the connection minus the number of SDP messages received over the lifetime of the connection. A maximum of 65535 (2^{16}-1) buffers may be posted at any one time.

A4.3.1.4 LENGTH (LEN)

SDP message length in bytes.

In a Hello or HelloAck message, the BSDH Len is equal to the sum of the sizes of the BSDH plus the HH or HAH, respectively. See sections A4.3.2.1 Hello Message (HH) on page 1204 and A4.3.2.2 HelloAck Message (HAH) on page 1208.

CA4-10: For SDP messages other than Hello and HelloAck, Len **shall** be equal to the sum of the sizes of the BSDH, extended header (if present), and ULP payload (if present). The IB message length **shall** equal the value of Len.

A4.3.1.5 MESSAGE SEQUENCE NUMBER (MSEQ)

CA4-11: The first SDP message sent after connection establishment **shall** have a MSeq value of 1. Each successive SDP message **shall** increase the value of Mseq by one, wrapping to zero after 0xFFFFFFFF.

A4.3.1.6 MESSAGE SEQUENCE NUMBER ACKNOWLEDGEMENT (MSEQACK)

MSeqAck is the sequence number of the last SDP message received by the local peer. See section A4.6.5.4 SrcAvail Revocation on page 1240 for additional constraints.

CA4-12: If no messages have been received by the local peer since connection establishment, a MSeqAck value of zero **shall** be transmitted.

A4.3.2 CONNECTION MANAGEMENT MESSAGES

SDP connection setup uses InfiniBand CM REQ, REP, and RTU MADs, with additional SDP information exchanged in the private data. SDP specific information includes the size of the receive private buffers, the initial credits for the receive private buffers, and source and destination IP addresses, as well as other parameters. SDP connection teardown uses InfiniBand DREQ and DREP messages, plus additional SDP messages (DisConn and AbortConn) to emulate TCP connection teardown semantics.

A4.3.2.1 HELLO MESSAGE (HH)

The Hello message shall contain a BSDH and a Hello Header (HH). The Hello message shall be contained in the private data of the CM REQ MAD. See section A4.5.1 Connection Setup on page 1218.

CA4-13: The Hello Header format **shall** be as defined in Figure 252.

Figure 252 Hello Header

bits bytes	31-24		23-16		15-8	7-0
0-3	MajV	MinV	IPV	Cap	rsvd	MaxAdverts
4-7	DesRemRcvSz					
8-11	LocalRcvSz					
12-15	LocalPort				rsvd	
16-19	SrcIP(127-96)					
20-23	SrcIP(95-64)					
24-27	SrcIP(63-32)					
28-31	SrcIP(31-00)					
32-35	DstIP(127-96)					
36-39	DstIP(95-64)					
40-43	DstIP(63-32)					
44-47	DstIP(31-00)					
48-51	rsvd				ExtMaxAdverts	
52-55	rsvd					
...						
72-75						

CA4-14: The BSDH fields of the REQ MAD **shall** be set as follows:

- MID = Hello
- len = size of the BSDH, plus the size of the HH
- flags = 0
- bufs = see section A4.5.1 Connection Setup on page 1218 and section A4.7.3 Initialization of Send Credit on page 1245.
- MSeq = set to zero and not checked on receive.
- MSeqAck = set to zero and not checked on receive.

A4.3.2.1.1 MAJOR PROTOCOL VERSION NUMBER (MAJV) - 4 BITS

The current specification requires MajV to be set to 2. See section A4.5.1 Connection Setup on page 1218 for additional information.

CA4-15: The accepting peer **shall** reject the connection if MajV in the HH does not match its local value.

A4.3.2.1.2 MINOR PROTOCOL VERSION NUMBER (MINV) - 4 BITS

The current specification requires MinV to be set to 2.

CA4-16: The accepting peer **shall not** reject the connection, solely on the basis that MinV of the HH does not match its local value.

This enables future protocol extensions which are upwardly compatible.

A4.3.2.1.3 IP VERSION (IPV) - 4 BITS

The Internet Protocol version number of the end-point address field (fields SrcIP and DstIP). If IPV = 0x4, the IP addresses are IP version 4 format (32 bit addresses). If IPV = 0x6, then both IP addresses are IP version 6 format (128 bit addresses). All other IPV values are reserved

CA4-17: The accepting peer **shall** reject the connection if IPV of the HH has a value other than 0x4 or 0x6.

A4.3.2.1.4 MAXIMUM ADVERTISEMENTS (MAXADVERTS) - 8 BITS

The maximum number of concurrent Zcopy advertisements that can be outstanding to the local QP at any one time. This includes SrcAvail advertisements for data transfer from the remote peer to the local peer and SinkAvail advertisements for data transfer from the local peer to the remote peer. MaxAdverts may be between 1 and 2^8-1, inclusive.

CA4-18: The accepting peer **shall** reject the connection if MaxAdverts of the HH is zero.

Note that there is no correlation between this parameter and InfiniBand RDMA Read resources (the Responder Resources parameter specified during connection setup).

A4.3.2.1.5 DESIRED REMOTE RECEIVE SIZE (DESREMRCVSZ) - 32 BITS

Desired size of remote peer's receive private buffers, in units of bytes (maximum = 2^{31} bytes). Usually set to the initial size of the local send buffers (assuming the send buffers are all the same size). This is a hint to the remote peer. The remote peer should take this into consideration when choosing the size of its receive private buffers, but it is free to select a different size.

A4.3.2.1.6 LOCAL RECEIVE SIZE (LOCALRCVSZ) - 32 BITS

Initial size of the local receive private buffers, in units of bytes (maximum = 2^{31} bytes).

A4.3.2.1.7 LOCALPORT - 16 BITS

The local TCP port number. See IETF RFC 1122 (Host Requirements RFC) for requirements on the TCP port number.

A4.3.2.1.8 INTERNET PROTOCOL ADDRESS (SRCIP, DSTIP) - 128 BITS

The Internet Protocol (IP) address for the local interface (SrcIP) and remote interface (DstIP). These can be either IPv4 or IPv6 addresses, as specified by the IPV field.

CA4-19: If SrcIP and DstIP of the HH are IPv4 addresses, as specified by the IPV field, then SrcIP(31-0) and DstIP(31-0) **shall** be used to transmit the source and destination IP addresses, respectively, and bits SrcIP(127-32) and DstIP(127-32) **shall** be set to zero.

A4.3.2.1.9 RSVD

Reserved for future use. Must be transmitted as zeroes and not checked on receive.

A4.3.2.1.10 CAPABILITIES (CAP) - 4 BITS

The Capabilities field conveys the connecting peer's capabilities. The following capabilities are defined.

CA4-19.2.1: SDP Extensions Table 353 "Capabilities" shall be used to define the Hello Header Cap field.

Table 353 Capabilities

Bit Position	Capability	*Description*
0	INVALIDATE_CAP	Supports incoming *Send w/Invalidate opcode*
1	EXTENDED_MAXADVERTS	*Extended MaxAdverts* is used
2-3	reserved	transmitted as zero and not checked at receiver

CA4-19.2.2: The connecting peer shall set the *INVALIDATE_CAP* bit in the *Cap* field in *Hello Header* only if it can support incoming invalidate messages.

If the accepting peer does not support *Send w/Invalidate,* it will ignore the *Invalidate Capability* advertisement.

CA4-19.2.3: The following rules shall be applied by a connecting peer when using *EXTENDED_MAXADVERTS* capability bit:

- Set the *EXTENDED_MAXADVERTS* bit only if the connecting peer supports more than 255 outstanding Zcopy advertisements on the local QP.
- If *EXTENDED_MAXADVERTS* bit is set, the *MaxAdverts* field shall be set to 255.

- If *EXTENDED_MAXADVERTS* bit is set, the *Extended MaxAdverts* field shall have the maximum number of outstanding Zcopy advertisements that the connecting peer supports.

A4.3.2.1.11 EXTENDED MAXADVERTS (EXTMAXADVERTS) - 16 BITS

The maximum number of concurrent Zcopy advertisements that can be outstanding to the local QP at any time. If EXTENDED_MAXADVERTS bit is set, ExtMaxAdverts must be between 256 and 2^{16}-1, inclusive. If EXTENDED_MAXADVERTS bit is clear, ExtMaxAdverts must be zero.

A4.3.2.2 HELLOACK MESSAGE (HAH)

The HelloAck (Hello Acknowledgement) message shall contain a BSDH and a HelloAck Header (HAH). The HelloAck message contains a subset of the information sent in the Hello message. The HelloAck message shall be contained in the private data of the CM REP MAD. See section A4.5.1 Connection Setup on page 1218

CA4-20: The HelloAck Header format **shall** be as defined in Figure 253.

Figure 253 HelloAck Header

bits / bytes	31-24		23-16		15-8	7-0
0-3	MajV	MinV	rsvd	Cap	ExtMaxAdverts	
4-7	ActRcvSz					
8-11	rsvd					
...						
176-179						

CA4-21: The BSDH fields of the REP MAD **shall** be set as follows:

- MID = HelloAck
- len = size of the BSDH, plus the size of the HAH
- flags = 0
- bufs = see section A4.5.1 Connection Setup on page 1218 and section A4.7.3 Initialization of Send Credit on page 1245.
- MSeq = set to zero and not checked on receive.
- MSeqAck = set to zero and not checked on receive.

A4.3.2.2.1 MAJOR PROTOCOL VERSION NUMBER (MAJV) - 4 BITS

The current specification requires MajV to be set to 2. See section A4.5.1 Connection Setup on page 1218 for additional information.

CA4-22: The connecting peer **shall** terminate the connection attempt if MajV does not match its local value, i.e., it sends a REJ back to the remote peer, instead of RTU.

A4.3.2.2.2 MINOR PROTOCOL VERSION NUMBER (MINV) - 4 BITS

The current specification requires MinV to be set to 2.

CA4-23: The connecting peer **shall not** terminate the connection attempt, solely on the basis that MinV of the HAH does not match its local value.

A4.3.2.2.3 EXTENDED MAXADVERTS (EXTMAXADVERTS) - 16 BITS

The maximum number of concurrent Zcopy advertisements that can be outstanding to the local peer at any one time. This includes SrcAvail advertisements sent to the local peer for data transfer from the remote peer to the local peer and SinkAvail advertisements for data transfer from the local peer to the remote peer.

CA4-24: This compliance statement has been obsoleted.

CA4-24.2.1: If the connecting peer has set *MajV* of 2 and *MinV* of 1 in the *HelloHeader*, the valid values for the *ExtMaxAdverts* in *HAH* are between 1 and 2^8-1, inclusive.

CA4-24.2.2: If the connecting peer has set *MajV* of 2 and *MinV* of 2 in the *HelloHeader*, the valid values for the *ExtMaxAdverts* in *HAH* are between 1 and 2^{16}-1, inclusive.

CA4-24.2.3: The connecting peer **shall** terminate the connection attempt if *ExtMaxAdverts* of the HAH is set to zero.

Note that there is no correlation between this parameter and InfiniBand RDMA Read resources (the Responder Resources parameter specified during connection setup).

A4.3.2.2.4 ACTUAL RECEIVE SIZE (ACTRCVSZ) - 32 BITS

The initial size of the local receive private buffers, in units of bytes (maximum = 2^{31} bytes).

A4.3.2.2.5 RSVD

Reserved for future use. Must be transmitted as zeroes and not checked on receive.

A4.3.2.2.6 CAPABILITIES (CAP) - 4 BITS

The Capabilities field conveys the accepting peer's capabilities. The following capabilities are defined.

CA4-24.2.4: SDP Extensions Table 354 "Capabilities" shall be used to define the HelloAck Header Cap field.

Table 354 Capabilities

Bit Position	Capability	*Description*
0	INVALIDATE_CAP	Supports incoming *Send w/Invalidate opcode*
1-3	reserved	transmitted as zero and not checked at receiver

CA4-24.2.5: The accepting peer shall set the *INVALIDATE_CAP* bit in the *Cap* field in *HelloAck Header* only if it can support incoming invalidate messages and the incoming *Hello Header Cap* field has the INVALIDATE_CAP bit set.

If both connecting and accepting peers in an SDP connection set the *INVALIDATE_CAP* bit in the *Cap* field of *Hello Header* and *HelloAck Header* respectively, the SDP connection is configured to be *Invalidate Enabled.*

A4.3.2.3 DISCONN MESSAGE

The DisConn (Disconnect Connection) message informs the remote peer that the local ULP will not be sending any more data on this connection, and that the ULP has requested graceful teardown of the socket in the send direction. This is functionally equivalent to TCP sending a FIN packet. See section A4.5.3 Connection Teardown on page 1223.

The DisConn message shall have only a BSDH. It contains no ULP payload.

A4.3.2.4 ABORTCONN MESSAGE

The AbortConn (Abort Connection) message tells the remote peer to ignore an earlier DisConn message and to consider socket teardown as abortive. This message is sent only if a DisConn message has been sent earlier. AbortConn is functionally equivalent to TCP setting the RST bit to reset a connection. See section A4.5.3 Connection Teardown on page 1223.

The AbortConn message shall have only a BSDH. It contains no ULP payload.

A4.3.3 DATA TRANSFER AND FLOW CONTROL MESSAGES

A4.3.3.1 DATA MESSAGE

The Data message is normally used to send ULP payload. A Data message without ULP payload is a gratuitous credit update.

The Data message is one of three SDP message types which may contain ULP data. The other types are SrcAvail and SinkAvail messages. See section A4.6.1 Bcopy on page 1228.

The Data message shall contain a BSDH.

A4.3.3.2 SRCAVAIL MESSAGE (SRCAH)

The SrcAvail (Data Source Available) message is sent by the Data Source to the Data Sink to inform the latter of the availability of an RDMA buffer that can be transferred through an RDMA Read operation.

The SrcAvail message shall contain a BSDH and a SrcAvail Header (SrcAH).

The SrcAH may include a copy of the initial portion of the send RDMA buffer as ULP payload of the SDP message, depending upon the flow control mode. See section A4.6.2 Read Zcopy on page 1229).

CA4-25: The SrcAvail Header format **shall** be as defined in Figure 254.

Figure 254 SrcAvail Header (SrcAH)

bits bytes	31-24	23-16	15-8	7-0
0-3	Len			
4-7	R_Key			
8-11	VA (63-32)			
12-15	VA (31-0)			

A4.3.3.2.1 LENGTH (LEN) - 32 BITS

The size of the send RDMA buffer (in bytes) represented by the VA and R_Key.

CA4-26: The value of Len **shall** be greater than zero and less than or equal to 2^{31} bytes.

A4.3.3.2.2 VIRTUAL ADDRESS (VA) - 64 BITS

The start address of the send RDMA buffer. The RDMA VA may start on any byte boundary.

CA4-27: The buffer addressed by the RDMA VA **shall** include the initial ULP data that was copied into the ULP payload of the SrcAvail message (if any).

A4.3.3.2.3 R_KEY - 32 BITS

CA4-28: The R_Key field **shall** contain the R_Key value to be used by the Data Sink, when retrieving data from the memory of the Data Source via an RDMA Read.

A4.3.3.3 SINKAVAIL MESSAGE (SINKAH)

The SinkAvail (Data Sink Available) message is sent by the Data Sink to the Data Source to inform the latter of the availability of an RDMA buffer that can be filled through an RDMA Write operation. See section A4.6.3 Write Zcopy on page 1233.

CA4-29: The SinkAvail Header format **shall** be as defined in Figure 255.

Figure 255 SinkAvail Header (SinkAH)

bits / bytes	31-24	23-16	15-8	7-0
0-3	Len			
4-7	R_Key			
8-11	VA (63-32)			
12-15	VA (31-0)			
16-19	NonDiscards			

The SinkAvail message shall contain a BSDH and SinkAH.

The SinkAvail message may include some send data as ULP payload for data flow in the *opposite* direction. See section A4.6.4 Transaction Mechanism on page 1235.

A4.3.3.3.1 LENGTH (LEN) - 32 BITS

The size (in bytes) of the receive RDMA buffer represented by the VA and R_Key.

CA4-30: The value of Len **shall** be greater than zero and less than or equal to 2^{31} bytes.

A4.3.3.3.2 VIRTUAL ADDRESS (VA) - 64 BITS

The start address of the receive RDMA buffer. The RDMA VA may start on any byte boundary.

A4.3.3.3.3 R_KEY - 32 BITS

CA4-31: The R_Key field **shall** contain the R_Key to be used by the Data Source, when sending the data via an RDMA Write into the memory of the Data Sink.

A4.3.3.3.4 NONDISCARDS - 32 BITS

The NonDiscards field in the SinkAvail message contains the Data Sink's current local value for NonDiscards. After connection setup, the Data Sink shall initialize the local value for NonDiscards to zero. The Data Sink shall increment its local value for NonDiscards when a ULP payload carrying SDP message is received and the SDP message did not cause the Data Sink to discard a previously sent SinkAvail message. This count wraps around to 0 after reaching 0xFFFFFFFF.

See section A4.6.5.1 Detecting Stale SinkAvail Advertisements on page 1237 for additional information.

A4.3.3.4 RDMA MESSAGES

SDP uses the existing InfiniBand RDMA Write and RDMA Read messages to transfer zero-copy data.

A4.3.3.5 SENDSM MESSAGE

The Data Source uses a SrcAvail message to inform the Data Sink of data which can be transferred using RDMA. If the Data Sink is unable or unwilling to transfer this data using RDMA, it can use the SendSm (Send Small) message to inform the Data Source it should send the data using the Bcopy Transfer Mechanism. See section A4.6.5.2 Mechanisms For Forcing Bcopy on page 1238.

The SendSm message shall contain only a BSDH and no ULP payload.

A4.3.3.6 RDMAWRCOMPL MESSAGE (RWCH)

The RdmaWrCompl (RDMA Write Complete) message is sent by the Data Source to inform the Data Sink of completion of an RDMA Write transfer. See section A4.6.3 Write Zcopy on page 1233.

CA4-32: The RdmaWrCompl Header format **shall** be as defined in Figure 256.

Figure 256 RdmaWrCompl Header (RWCH)

bits bytes	31-24	23-16	15-8	7-0
0-3	Len			

The RdmaWrCompl message shall contain only a BSDH and a RWCH.

A4.3.3.6.1 LENGTH (LEN) - 32 BITS

The size (in bytes) transferred to the receiver's RDMA buffer through RDMA Write(s) for the oldest outstanding SinkAvail. The size may be less than the size of the RDMA buffer advertised by the Data Sink in the SinkAvail message.

A4.3.3.7 RDMARDCOMPL MESSAGE (RRCH)

The RdmaRdCompl (RDMA Read Complete) message is sent by the Data Sink to inform the Data Source of completion of an RDMA Read transfer. See section A4.6.2 Read Zcopy on page 1229

CA4-33: The RdmaRdCompl Header format **shall** be as defined in Figure 257.

Figure 257 RdmaRdCompl Header (RRCH)

bits bytes	31-24	23-16	15-8	7-0
0-3	Len			

The RdmaRdCompl message shall contain only the BSDH and RRCH.

The sender (Data Sink) may set the REQ_PIPE bit in the Flags field of the BSDH. This bit should be set to zero if the Data Sink would prefer the Data Source to stay in Combined Mode. The REQ_PIPE bit should be set to one if the Data Sink prefers the Data Source to switch to Pipelined Mode or remain in Pipelined Mode. This is just a hint to the Data Source. The Data Source is not obligated to follow the recommendation. The Data Source (which is the receiver of the RdmaRdCompl message) does not examine the REQ_PIPE bit when in Buffered Mode.

A4.3.3.7.1 LENGTH (LEN) - 32 BITS

The size (in bytes) transferred to the Data Sink's RDMA buffer from the Data Source's RDMA buffer (i.e., the buffer advertised in the original SrcAvail message) through RDMA Read(s). The Len field does not include the portion of the buffer (if any) transferred within the SrcAvail message as ULP data payload. If there is ULP data in the SrcAvail message, Len shall be less than the size of the Data Source RDMA buffer advertised in the SrcAvail message by exactly the number of ULP payload bytes included in the SrcAvail message.

A4.3.3.8 MODECHANGE MESSAGE (MCH)

The ModeChange message is used to inform the remote peer of a flow control mode transition. See section A4.8 SDP Modes on page 1248.

CA4-34: The ModeChange Header format **shall** be as defined in Figure 258.

Figure 258 ModeChange Header (MCH)

bits bytes	31-24		23-16	15-8	7-0
0-3	S	Mode	RSVD		

The ModeChange message shall contain only a BSDH and MCH.

The receiver of the ModeChange message must change its send or receive mode to the new mode specified in the message.

A4.3.3.8.1 S - 1 BIT

Specifies whether the peer receiving the ModeChange message should change its flow control mode for its Send half-connection (S = 1) or Receive half-connection (S = 0), for the connection over which the ModeChange message was received.

A4.3.3.8.2 MODE - 3 BITS

Specifies the new mode.

CA4-35: The ModeChange MCH field value **shall** be as defined in Table 355.

Table 355 MCH Mode Values

Mode Value	Name	Description
0	BUFF_MODE	New mode should be Buffered Mode
1	COMB_MODE	New mode should be Combined Mode
2	PIPE_MODE	New mode should be Pipelined Mode
3-7	reserved	Reserved value

A4.3.3.8.3 RSVD - 28 BITS

Reserved for future use. Must be transmitted as zeroes and not checked on receive.

A4.3.3.9 SRCAVAILCANCEL MESSAGE

The SrcAvailCancel (Data Source Available Cancel) message is sent by the Data Source to ask the Data Sink to ignore all SrcAvail advertisements sent by the Data Source which are Unprocessed by the Data Sink. See section A4.6.5.4 SrcAvail Revocation on page 1240.

The SrcAvailCancel message shall contain only a BSDH.

A4.3.3.10 SINKAVAILCANCEL MESSAGE

The SinkAvailCancel (Data Sink Available Cancel) message is sent by the Data Sink to ask the Data Source to ignore all SinkAvail advertisements sent by the Data Sink which are Unprocessed by the Data Source. See section A4.6.5.5 SinkAvail Revocation on page 1242.

The SinkAvailCancel message shall contain only a BSDH.

A4.3.3.11 SINKCANCELACK MESSAGE

The SinkCancelAck (Data Sink Available Cancel Acknowledgement) message is sent by the Data Source in response to the SinkAvailCancel message. It shall be sent after it has canceled all Unprocessed SinkAvail advertisements. See section A4.6.5.5 SinkAvail Revocation on page 1242.

The SinkCancelAck message shall contain only a BSDH.

A4.3.4 PRIVATE BUFFER RESIZING MESSAGES

A4.3.4.1 CHRCVBUF MESSAGE (CRBH)

The ChRcvBuf (Change Receive private Buffer size) message is sent by a the Data Source to the Data Sink to request a change in the size of the latter's receive private buffers. See section A4.7.6 Receive Buffer Resizing on page 1247.

The ChRcvBuf message shall contain only a BSDH and a CRBH

CA4-36: The ChRcvBuf Header format **shall** be as defined in Figure 259.

Figure 259 ChRcvBuf Header (CRBH)

bits bytes	31-24	23-16	15-8	7-0
0-3	DesSz			

A4.3.4.1.1 DESIRED SIZE (DESSZ) - 32 BITS

Desired size (in bytes) of the Data Sink's receive private buffers.

A4.3.4.2 CHRCVBUFACK MESSAGE (CRBAH)

The ChRcvBufAck (Change Receive private Buffer size Acknowledgement) message is sent in response to the ChRcvBuf message. The ChRcvBufAck message informs the Data Source of the new size of the receive private buffers. See section A4.7.6 Receive Buffer Resizing on page 1247.

The ChRcvBufAck message shall contain only a BSDH and a CRBAH.

CA4-37: The ChRcvBufAck Header format **shall** be as defined in Figure 260.

Figure 260 ChRcvBufAck Header (CRBAH)

bits / bytes	31-24	23-16	15-8	7-0
0-3	ActSz			

A4.3.4.2.1 ACTUAL SIZE (ACTSZ) - 32 BITS

The actual or new size (in bytes) of the local receive private buffers. The actual size may be the same as the size prior to receipt of the ChRcvBuf message if the protocol implementation does not wish to resize its receive private buffers.

A4.3.5 SOCKET DUPLICATION MESSAGES

A4.3.5.1 SUSPCOMM MESSAGE

The SuspComm (Suspend Communication) message is sent to ask the remote peer to suspend communication as part of preparing the socket for duplication. See section A4.10 Socket Duplication on page 1260.

The SuspComm message shall contain only a BSDH and a SuspCH.

CA4-38: The SuspComm Header format **shall** be as defined in Figure 261.

Figure 261 SuspComm Header (SuspCH)

bits / bytes	31-24	23-16	15-8	7-0
0-3	Service ID(63-32)			
4-7	Service ID(31-0)			

A4.3.5.1.1 SERVICE ID - 64 BITS

The Service ID that the remote peer should try to connect with to re-establish the connection with the local peer after duplication has completed.

A4.3.5.2 SUSPCOMMACK MESSAGE

The SuspCommAck (Suspend Communication Acknowledgement) message is sent in response to the SuspComm message. This message informs the peer that all communication has been suspended as requested by the peer in its SuspComm message. See section A4.10 Socket Duplication on page 1260.

The SuspCommAck message shall contain only a BSDH.

A4.4 ADDRESS RESOLUTION

CA4-39: SDP **shall** use either IPv4 or IPv6 addressing (generically called IP addressing), as defined in IETF RFC791 and IETF RFC2460, respectively. SDP **shall** use the IP over InfiniBand address resolution mechanism. This mechanism is being developed by the IP over IB (IPoIB) IETF working group. See http://www.ietf.org/html.charters/ipoib-charter.html.

SDP does this rather than defining a new mechanism for mapping from an IP address to an InfiniBand address (either a LID or GID). Thus the SDP protocol definition begins after the source and destination addresses have been resolved to an InfiniBand address.

IP over InfiniBand does not define a mechanism to perform an inverse lookup (from an InfiniBand address to an IP address). It is also possible for a single InfiniBand address to have many IP addresses, providing a one-to-many mapping when attempting to perform an inverse lookup. To resolve these issues, the complete source and destination IP address is provided during connection setup to enable mapping the destination and source LID/GID to an IP address at the accepting peer of the connection.

A4.5 CONNECTION MANAGEMENT

SDP connection setup uses the existing InfiniBand connection management MADs (REQ/REP/RTU/REJ/MRA), which include the ability to redirect the connection to a different CM or Port (see section 12.10.7 Active Client to Passive Server with Redirector - All Accept Communication on page 704) or to automatically migrate the connection to an alternate path (APM, see sections 10.4 Automatic Path Migration on page 461 and 17.2.8 Automatic Path Migration on page 1031). SDP connection teardown uses the existing InfiniBand teardown messages (DREQ/DREP), but requires additional SDP message types to emulate TCP connection teardown semantics for abortive and graceful connection teardown.

A4.5.1 CONNECTION SETUP

A4.5.1.1 INFINIBAND RELIABLE CONNECTION SETUP

CA4-40: SDP communications **shall** use InfiniBand's Reliable Connection transport service exclusively.

Recall that an SDP implementation uses the private data field in the InfiniBand CM REQ and CM REP MADs to transfer initial connection information through the Hello and HelloAck messages, respectively.

Each socket corresponds to a single queue pair. The CM REQ MAD also contains the destination Service ID, which is used to map the connection to the destination TCP port number. The connection setup mechanism is the InfiniBand Active/Passive (Client/Server) model as specified in section 12.2 Establishment on page 652.

CA4-41: The connection setup sequence **shall** include the following numbered steps in order:

4) The connecting peer prepares to send a CM REQ MAD with private data.
   a) Socket applications use 16-bit TCP port numbers to establish connections. InfiniBand requires a 64-bit Service ID for connection setup. The mapping between TCP port numbers and InfiniBand Service IDs is defined in the Application Specific Identifiers Annex. It is reproduced here for convenience, where the TCP port number is a hex value, 0xXXXX.

      Service ID = 0x0000 0000 0001 XXXX

   b) The connecting peer configures how many local receive private buffers will be posted to the HCA and the size of the receive private buffers. See section A4.7 Private Buffer Management on page 1244 for constraints on private buffers.
   c) The connecting peer should post all of its receive private buffers to the Receive Queue (using the Post Receive Request verb) at this time or before sending the CM RTU MAD. The connecting peer may wait until immediately after completion of connection setup to post the buffers, however it is possible the InfiniBand RNR NAK protocol will be invoked. The RNR NAK protocol will stall data transfer until the receive private buffers are posted. See section A4.7 Private Buffer Management on page 1244.
   d) The Responder Resources field in the CM REQ MAD **shall** be greater than or equal to one.
5) The connecting peer **shall** send a CM REQ MAD with the Hello message to the accepting peer. See section A4.3.2.1 Hello Message (HH) on page 1204 and section A4.7.3 Initialization of Send Credit on page 1245 for additional information on filling in the REQ private data.
6) The accepting peer, upon receipt of the Hello message, performs the following operations:
   a) The accepting peer should post all of its receive private buffers to the Receive Queue (using the Post Receive Request verb) before sending the CM REP MAD with the HelloAck message. The accepting peer may wait until immediately after completion of connection setup to post the buffers, however it is possible the InfiniBand RNR NAK protocol will be invoked. The RNR NAK Protocol will stall data transfer until the receive private buffers are posted. See section A4.7 Private Buffer Management on page 1244.
   b) The Responder Resources field in the CM REP MAD **shall** be greater than or equal to one.

c) The accepting peer **shall** reject the connection by sending a CM REJ MAD if:

- The local Major Protocol Version Number does not match the Major Protocol Version Number in the Hello message. If the accepting peer's Major Protocol Version Number matches, but the Minor Protocol Version Number does not, the accepting peer **shall** accept the connection. The protocol specified by the lower Minor Protocol Version Number **shall** be used for all communication after connection setup.
- If the Responder Resources field in the CM REQ MAD is equal to zero.

7) If the connection is accepted, the accepting peer **shall** send a CM REP MAD with a HelloAck message back to the connecting peer. See section A4.3.2.2 HelloAck Message (HAH) on page 1208 for additional information.

8) The connecting peer replies to the CM REP MAD HelloAck message with a CM RTU MAD.

a) If the connecting peer did not previously post receive private buffers to the receive queue, it should do so before sending the CM RTU MAD. It may wait until immediately after the CM RTU MAD is sent, but as previously mentioned, this may stall data transfer until the buffers are posted.

b) The connecting peer **shall** set the flow control mode to Combined Mode and may immediately commence data transfer.

c) The connecting peer **shall** reject the connection by sending a CM REJ MAD if:

- The local Major Protocol Version Number does not match the Major Protocol Version Number in the HelloAck message.
- The Responder Resources field in the CM REP MAD is equal to zero.

9) The accepting peer receives the CM RTU MAD.

a) The accepting peer **shall** set the flow control mode to Combined Mode and may immediately commence data transfer.

If the RTU is dropped, the accepting peer may implement a timeout and retransmit the REP. If a retransmission timeout is not implemented, it is possible that the connecting peer's SDP messages may be received without an RTU having been delivered. This issue is resolved by the HCA generating an event which completes connection setup. However, if the accepting peer did not post receive private buffers before the SDP message was received, data transfer will stall until connection setup is completed and the receive private buffers are posted to the Receive Queue.

CA4-42: Implementations **shall** set the RNR NAK retry count to be non-zero to avoid teardown of the connection because the remote peer has not posted receive private buffers before completion of connection setup.

A4.5.1.2 ABORTING CONNECTION SETUP

CA4-43: When a CM REJ MAD is received by either the connecting or accepting peer the connection setup **shall** be aborted.

If a CM REJ MAD is sent for an SDP-specific error, the reject reason code value shall be 28 (Consumer Reject -- 12.6.7.2 Rejection Reason on page 665.

An SDP implementation is expected to clean up any resources associated with an aborted connection.

A4.5.2 AUTOMATIC PATH MIGRATION

SDP is designed to be compatible with the optional InfiniBand Automatic Path Migration feature defined in sections 10.4 Automatic Path Migration on page 461 and 17.2.8 Automatic Path Migration on page 1031 (APM). SDP implementations should support APM, if it is available on the underlying InfiniBand implementation. SDP consumers may at their option use this feature if it is available. APM usage involves two basic steps. First, alternate paths are determined. Second, those paths are configured into the SDP connection.

A4.5.2.1 DETERMINING ALTERNATE PATHS

Exactly how alternate paths are chosen is a matter of policy and outside the formal scope of the SDP specification. However, SDP was designed to be compatible with a number of different styles of alternate path selection. These styles include policies which use *a priori* information or information exchanged outside of the SDP protocol.

Alternate paths are commonly determined at connection setup time. However, they may be determined later. Also, new alternate paths may need to be determined later if fabric conditions change or to reload a new alternate path after path migration has occurred.

A4.5.2.2 EXAMPLE ALTERNATE PATH SELECTION PROCEDURE

The address resolution procedure (see section A4.4 Address Resolution on page 1218) provides address information in the form of a GID. If desired, implementations may use the following procedure to find an alternate path based on this address. The alternate path found by this procedure is not guaranteed to map back into a valid IP-on-IB path.

This alternate path procedure employs a query commonly used to obtain REQ information for the a single path.

SubnAdmGetTable(PathRecord: SGID, DGID, RawTraffic=0, other-parameters)

"other-parameters" may specify items such as SL, P_Key, MTU, Rate, etc.

Note that SDP does not require any particular MTU, while IP-on-IB paths have additional restrictions.

NumbPath could be 1 (in which case the SA query may use SubnAdmGet rather than SubnAdmGetTable) or it could be higher if some other criteria are used to select a path that cannot be specified in an SA query.

If an alternate path is desired to the above port, modify the single path query. One modification is to specify NumbPaths greater than 1 (up to 127) and choose two path records to use. (Note, it is possible that a fabric configuration may only provide a single possible path.) Another method is to repeat the single path query but specify different "other-parameters" (e.g. SL) to obtain an alternate path.

If it is desired that an alternate path be to another port of the HCA containing the above port, the following procedure may be used.

1) Obtain the primary path by the single path query described above.
2) Use the LID from the primary PathRecord to construct a RID (see section 15.2.4.2 Record Identifier (RID) Fields on page 887)

   (Note that section C15-0.1.9: on page 887 allows this to work even if the LID is not the base LID.)
3) Obtain the primary port NodeRecord (see section 15.2.5.2 NodeRecord on page 891) for the CA:

   SubnAdmGet(NodeRecord: RID)

   Save the NodeGUID, PortGUID and LocalPortNum fields from the NodeInfo (see section 14.2.5.3 NodeInfo on page 818) portion of this NodeRecord.
4) Get all other NodeRecords for this CA:

   SubnAdmGetTable(NodeRecord: NodeGUID)

   For each NodeRecord returned (there will be one per port), save the NodeRID. Also save the PortGUID and LocalPortNum from the NodeInfo portion. Note that one record returned from this query will be a duplicate of that returned from step 3. The duplicate can be found by matching on the LocalPortNum field.
5) For each possible alternate port, find its GID Prefix by querying the SA:

SubnAdmGet(PortInfoRecord: RID=NodeRecord.NodeRID,
LocalPortNum=NodeRecord.NodeInfo.LocalPortNum)

Save the GIDPrefix field from the PortInfoRecord.

6) Assemble the zeroth GID for each possible alternate port:

GID<127:64> = PortInfoRecord.PortInfo.GIDPrefix

GID<63:0> = NodeRecord.NodeInfo.PortGUID

7) Optionally, the other possible GIDs for each alternate port can be constructed by looking up GUIDInfoRecords. (This may be necessary in inter-subnet routing cases.)

8) The single path query can now be repeated on each possible alternate port using the GID(s) constructed in steps 6 and 7.

9) Select an alternate path from those looked up in step 8.

A4.5.2.3 CONFIGURING ALTERNATE PATHS

Configuration of alternate paths is accomplished using standard CM MADs. Two means are possible. One method is to specify the alternate path information in the CM REQ MAD as part of the normal REQ/REP/RTU exchange for connection setup. This procedure places the responsibility for specifying alternate paths with the connecting side. The second method uses the CM LAP MAD to modify an existing connection. Note that the LAP/APR MAD exchange can only be initiated from the connecting side.

The LAP can also be used to reconfigure alternate paths. If a two step process is again required, one LAP/APR sequence can exchange private data to drive a second LAP method configuration.

A4.5.3 CONNECTION TEARDOWN

SDP emulates TCP connection teardown functionality. TCP provides two ways to close a connection - a graceful close, where any data that has been posted by the ULP to the transport is transferred before the connection is torn down, and abortive close, where the connection is immediately torn down.

A4.5.3.1 GRACEFUL CLOSE

TCP's graceful close (also known as graceful disconnect or half-closed connections) is an agreement between the transport and ULP that:

- before the connection is terminated, all data accepted for transmission by the transport before the close occurred is guaranteed to be sent out (under reasonable limitations) and reliably acknowledged.

- data reception can continue normally until the remote peer performs a close.

Sockets Direct Protocol provides the same behavior.

CA4-44: The local peer **shall not** close the InfiniBand connection at the time of the ULP's call to gracefully close the half-connection. The local peer shall reject any send data posted by the ULP after the ULP close call occurred.

CA4-45: The local peer **shall** continue to receive ULP data through any of the SDP data transfer mechanisms until the remote peer gracefully closes the connection or the connection is abortively closed (see section A4.5.3.2 Abortive Close on page 1226 for the abortive close protocol).

CA4-46: The local peer **shall** also perform the following operations in the order specified:

1) The local peer **shall** complete the transmission of all outbound data posted by the ULP before the ULP requested the graceful close. This means that all Bcopy transfers, Write Zcopy transfers, Read Zcopy transfers, and Transaction transfers from this Data Source have been completed (see section A4.6 Data Transfer Mechanisms on page 1227). Completions may be successful or unsuccessful (e.g. an InfiniBand timeout occurred). Unsuccessful completions **shall** cause the local peer to perform an abortive close (see section A4.5.3.2 Abortive Close on page 1226).
2) The local peer **shall** send a DisConn message to the remote peer. This informs the remote peer that the connection is being terminated gracefully, allowing the remote peer to inform the ULP of this fact as appropriate. If the DisConn message completed with an error, the connection tear down was abortive.The DisConn message provides similar functionality to a TCP segment with the FIN bit set. Because InfiniBand reliable connection service provides in-order delivery, no ULP data will be received after the DisConn message has been received. Note that SDP messages may continue to be received by the local peer to enable ULP data transfer on the opposite half-connection.
3) The local peer **shall** wait for one of the following events:
   - reception of a DisConn message - the remote peer gracefully closed the opposite half-connection (unless an AbortConn message is received before the connection is terminated).
   - the InfiniBand connection is torn down (this may be due to either a graceful or an abortive close).
   - the local ULP abortively closes the connection (see section A4.5.3.2 Abortive Close on page 1226).

- if no forward progress is being made, the connection may be abortively closed (see section A4.5.3.2 Abortive Close on page 1226).

4) When the SDP implementation is informed that the InfiniBand connection was torn down all work completions **shall** be examined to determine whether a DisConn message, or a DisConn message followed by an AbortConn message, was received. If a DisConn message was received without an AbortConn, the graceful close was successfully completed. If a DisConn message was not received, or a DisConn message and an AbortConn message were received, then the close was abortive (see section A4.5.3.2 Abortive Close on page 1226). In either case, all ULP receive buffers **shall** be completed with information about how much of the buffer was filled.

5) The local peer **shall** wait for all send work requests to complete (either successfully or in error). This is necessary because the remote peer's DisConn message may have crossed the local peer's DisConn message (i.e. both sides are simultaneously attempting a graceful teardown). Because InfiniBand connection teardown immediately places the Queue Pair in the Error state (see section 12.10.8 Communication Release on page 705), any outstanding SDP messages or RDMA transfers would be aborted if the connection were immediately torn down.

6) The InfiniBand connection teardown protocol **shall** be used to complete the teardown (see section 12.10.8.1 Disconnect Request on page 705). The local peer **shall** close and clean up all InfiniBand resources associated with the connection (queue pair, buffers, etc.).

CA4-47: To effect a graceful close, the remote peer **shall** perform the following operations in the order given:

1) Upon receipt of a DisConn message, the remote peer **shall** consider all its outstanding SinkAvail advertisement as canceled, complete all ULP receive buffers, and wait for the ULP to close the connection. The remote peer **shall** continue to allow normal ULP send data transfer, but **shall** complete any new ULP receive buffers and inform the ULP (as appropriate) that the receive half-connection has been gracefully closed.

2) If the ULP issues an abortive close, the abortive close protocol **shall** be used (see section A4.5.3.2 Abortive Close on page 1226). If the ULP issues a graceful close, the remote peer **shall** complete the transmission of all send ULP data that was posted before the ULP posted the graceful close. Completions may be successful or unsuccessful (e.g. an InfiniBand timeout occurred). The remote peer **shall** reject any send data posted by the ULP after the ULP close call occurred.

3) The remote peer **shall** send a DisConn message to the local peer.

4) The remote peer **shall** wait for all send work requests to complete and then **shall** process all work completions. If the DisConn message completed without error and no AbortConn message was received, then the graceful teardown was successful. If the DisConn message completed with an error (including a flush error status) or an AbortConn was received, the connection teardown was abortive.

5) The InfiniBand connection teardown protocol **shall** be used to complete the teardown. The remote peer **shall** close and clean up all InfiniBand resources associated with the connection (queue pair, buffers, etc.).

A4.5.3.2 ABORTIVE CLOSE

CA4-48: If the ULP specifies an abortive disconnect or an abortive disconnect is required for some other reason, the following rules **shall** be followed:

- if the IB connection is still valid and a DisConn message was previously sent, then an AbortConn message **shall** be sent and its completion processed before terminating the InfiniBand connection.
- if the IB connection is not valid or if the IB connection is valid and no DisConn message was previously sent, no further SDP messages **shall** be placed on the Send Queue and the InfiniBand connection **shall** be terminated immediately (see section 12.10.8 Communication Release on page 705).

CA4-49: SDP **shall** consider the connection abortively torn down if the InfiniBand connection is torn down without receiving a DisConn message, or if both an AbortConn and a DisConn message were received. Any unsent ULP data **shall** be discarded.

If an SDP protocol violation occurs, the connection should be abortively closed. A protocol violation includes but is not limited to InfiniBand errors, invalid SDP messages, or incorrectly formatted SDP messages.

Certain ULP behaviors can lead to a situation under which the ULP initially indicates graceful teardown in the send direction (causing the DisConn message to go out), and then some error occurs which requires the connection to be abortively closed. The AbortConn message is used for this purpose. It is sent if the DisConn message has already been sent out (but InfiniBand connection has not been terminated yet) and some error condition arises which calls for abortive teardown of the socket under TCP semantics. Sending out the AbortConn message informs the remote peer to ignore the earlier DisConn message and inform the ULP (as appropriate) that the connection was closed abortively. In this case, the AbortConn message is similar to TCP sending a segment with the RST bit set after it has already sent a segment with the FIN bit set.

CA4-50: Once the AbortConn message completion event occurs, the normal InfiniBand connection teardown protocol **shall** be used to complete the teardown.

A4.6 DATA TRANSFER MECHANISMS

SDP employs four data transfer mechanisms:

- Bcopy - transfer of ULP data from send buffers into receive private buffers.
- Read Zcopy - transfer of ULP data through RDMA Reads, preferably directly from ULP buffers into ULP buffers.
- Write Zcopy - transfer of ULP data through RDMA Writes, preferably directly from ULP buffers into ULP buffers.
- Transaction - an optimized ULP data transfer model for transactions. It piggy-backs ULP data transfer using private buffers on top of the Write Zcopy mechanism used to transfer ULP data on the opposite half-connection.

The policy which controls when to use the Bcopy data transfer mechanisms versus a Zcopy data transfer mechanism is outside the scope of this specification. An implementation dependent parameter defined as the Bcopy Threshold is used to abstractly define the results of the policy decision. There are no constraints placed on the Data *Source* Bcopy Threshold values, and the value of the Data Source Bcopy Threshold may be static or dynamic. The Data *Sink* Bcopy Threshold has a single constraint: it must be greater than or equal to the size of the receive private buffers. Its value may also be static or dynamic.

Note that some socket implementations do not provide deterministic results if overlapping receive buffers are posted.

CA4-51: An SDP implementation **shall** support the Bcopy data transfer mechanism, both as a Data Source and as a Data Sink.

It is strongly recommended that an SDP implementation support the ability to initiate all data transfer mechanisms.

It is strongly recommended that an SDP implementation support the ability to carry out all data transfer mechanism requests.

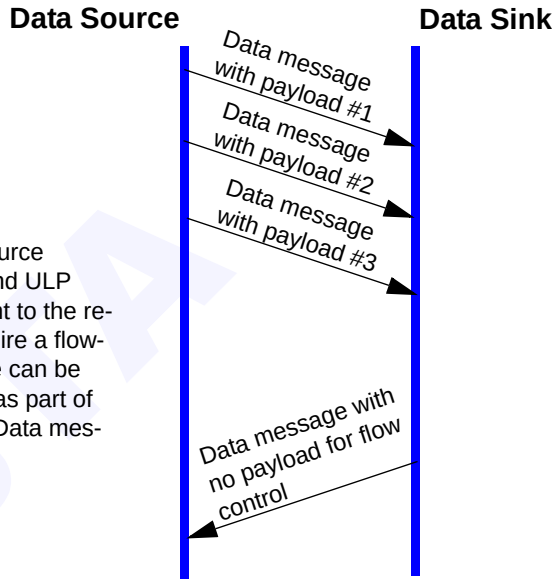
CA4-52: If an SDP implementation does not support carrying out a received request for a given optional data transfer mechanism, the implementation **must** still be able to parse the received request, and to force the use of the Bcopy data transfer mechanism.

For example, if an SDP implementation does not support carrying out the Read Zcopy data transfer mechanism request, when the implementation

receives a SrcAvail message, the implementation must be able to respond with a SendSm message to force the Data Source to use the Bcopy data transfer mechanism.

A4.6.1 BCOPY

SDP maintains a small set of receive private buffers for receiving data that the Data Source transfers using InfiniBand Sends. Each connection has a separate pool of receive private buffers.



A ladder diagram showing the Data Source sending multiple Data messages to send ULP data. Note that Data messages are sent to the receive private buffer pool, and thus require a flow-control update periodically. This update can be piggybacked on a SDP message sent as part of normal data flow or it can be sent in a Data message with no payload.

Figure 262 Ladder Diagram for BCopy Mechanism

Each peer chooses its own sizes of send and receive private buffers and informs the other peer of these during connection setup.

CA4-53: The Data Source **shall** limit the amount of ULP data sent in an SDP message (specifically a Data, SinkAvail, or SrcAvail message) to ensure the ULP data plus SDP header(s) will fit within the receive private buffer size advertised by the Data Sink.

SDP message transfer is flow controlled as described in section A4.7 Private Buffer Management on page 1244.

For the Data Source, data may be copied from the ULP's buffer to the payload sections of one or more of the send buffers or the ULP buffer may be referenced directly by the send work request. In the header, the MID is set to type Data message and the SDP message size is set to the ULP payload size plus the size of the header. The SDP message is sent by posting a Send work request.

The Data Sink receives the SDP message in its posted receive private buffers. When a ULP receive buffer is completed is outside the scope of this specification.

A4.6.2 READ ZCOPY

This mechanism shall transfer data through the following sequence of operations:

1) The Data Source sends a SrcAvail message when a send ULP buffer the Source deems suitable has been posted (there are no protocol restrictions on the Data Source use of the Bcopy mechanism versus Read Zcopy mechanism for transfer of a specific ULP buffer). For example a SrcAvail message may be sent if the ULP buffer is larger than the Bcopy Threshold. If the Source chooses to advertise a ULP buffer in a SrcAvail message, the ULP buffer is referred to as an RDMA buffer (the RDMA buffer may be a copy of the ULP buffer).

   Note that in Combined Mode the SrcAvail message payload must contain at least one byte of ULP payload and in Pipelined Mode the SrcAvail message must not contain ULP payload (see sections A4.8.2 Combined Mode on page 1251 and A4.8.3 Pipelined Mode on page 1251).

CA4-54: The SrcAH Len, VA, and R_Key fields **shall** describe the entire send RDMA buffer, regardless of whether a copy of the initial portion of the RDMA buffer is included in the SrcAvail message payload.

CA4-54.2.1: In an *Invalidate Enabled* SDP connection, the R_Key in a *SrcAvail* message **shall be** capable of being remotely invalidated. Refer to the *Base Memory Management Extensions* for additional details on remote invalidation.

   In an *Invalidate Enabled* SDP connection, the Data Source should not rely on the Data Sink to perform a remote invalidate after the data transfer is completed.

   After receiving a SrcAvail message, the Data Sink may send a SendSm message when ULP receive buffer(s) are not suitable for Read Zcopy.

2) The Data Sink receives the SrcAvail message and waits for the ULP to post a receive buffer to SDP. If the receive ULP buffer is viewed as unsuitable for Read Zcopy, a SendSm message should be sent (see section A4.6.5.2 Mechanisms For Forcing Bcopy on page 1238). If the receive ULP buffer is viewed as suitable for Read Zcopy, the ULP buffer should be used as the RDMA buffer. An implementation may choose to create an intermediate buffer as the RDMA buffer, and then copy the data into the ULP buffer. If there is an initial portion of the send RDMA buffer present in the SrcAvail advertisement, the

Data Sink shall move the data into the RDMA buffer through one of the following mechanisms:

- copy some or all of the ULP payload of the SrcAvail message to the receive RDMA buffer, and then perform one or more RDMA Read(s) to retrieve the rest of the data, offsetting the initial RDMA Read transfer by the number of bytes that were copied out of the SrcAvail message ULP payload.
- avoid the ULP payload copy and start the initial RDMA Read at the start of the send RDMA buffer. Additional RDMA Reads may be used to transfer the rest of the buffer.

CA4-55: After the RDMA Read(s) complete(s), the Data Sink **shall** send a RdmaRdCompl message to the Data Source, unless the operation was canceled (see section A4.6.5.4 SrcAvail Revocation on page 1240). The RdmaRdCompl **shall** either have the fence indicator set or the Data Sink **shall** wait for completion of the RDMA Read before posting the RdmaRdCompl to the Send Queue.

CA4-56: The RdmaRdCompl header **shall** contain the size (in bytes) of ULP data transferred through the RDMA Read(s), excluding any portion of the ULP data that was originally transferred through the SrcAvail message. The RdmaRdCompl message **shall** refer to data made available through a single SrcAvail advertisement.

CA4-57: A Data Sink **shall** only send an RdmaRdCompl message associated with the oldest incomplete SrcAvail message.

The size does not include the portion of the data, if any, transferred within the SrcAvail message as ULP payload. It may be less than the size of the Data Source RDMA buffer advertised in the SrcAvail message minus the size of ULP data payload included in the SrcAvail message. An implementation may loop performing a series of RDMA Read operations followed by RdmaRdCompl messages to transfer the send RDMA buffer contents.

It is expected (but not required) that protocol implementations would typically RDMA Read all the ULP data and then send a single RdmaRdCompl message to inform the Data Source that the SrcAvail message has been Processed. The facility to specify data transfer size less than the RDMA buffer size advertised in the SrcAvail message enables various transfer scenarios. For example, a protocol implementation may RDMA Read part of the data and then send a RdmaRdCompl message followed by a SendSm message to retrieve the rest of the data using the Bcopy mechanism. Only one SendSm message may be used to complete data transfer for a given SrcAvail advertisement. See 3) below.

In an *Invalidate Enabled* SDP connection, the Data Sink should send the last *RdmaRdCompl* message, associated with a *SrcAvail* mes-

sage, through a *Send w/Invalidate*; the R_Key used for the RDMA Read data transfer shall be used as an argument to the *Send w/Invalidate* verb.

CA4-57.2.1: The Data Sink shall not send an *RdmaRdCompl* message through a *Send w/Invalidate* for partially completed data transfers.

Note that if the flow control mode from the Data Source to the Data Sink is Combined Mode, the Data Sink may set the REQ_PIPE bit in the BSDH Flags field of the RdmaRdCompl message if it wishes to transition to Pipelined mode (see section A4.3.1 Base Sockets Direct Header (BSDH) on page 1201).

3) Upon receiving the RdmaRdCompl message, the Data Source shall compare the RRCH Len field against the length of the oldest, incomplete SrcAvail advertisement. If the send RDMA buffer has not been completely transferred, the Data Source shall wait until the SrcAvail message has been Processed by an ensuing RdmaRdCompl message or a SendSm message. Once the send RDMA buffer is completed, this may map to completion of a send ULP buffer, as appropriate.

CA4-57.2.2: If the Data Sink has remotely invalidated the RDMA buffer through a *RdmaRdCompl* message, the Data Source shall verify that the invalidated R_Key is associated with the RDMA buffer sent in the oldest incomplete *SrcAvail* message. If there is a mismatch, the Data Source shall view the operation as a protocol violation.

CA4-58: Data advertised by SrcAvail **shall** remain available for Read Zcopy by the Data Sink until its consumption has been acknowledged with RdmaRdCompl message(s), a SendSm message is received from the Data Sink, the SrcAvail message has been over ridden because of a SinkAvail message (see section A4.8.3 Pipelined Mode on page 1251), or the Data Source has Processed a SrcAvailCancel sequence (see section A4.6.5.4 SrcAvail Revocation on page 1240).

Note the portions of the RDMA Read buffer that have been completed by an RdmaRdCompl are no longer required to be available for RDMA Read.

If a SendSm message is received at the Data Source, the Data Source shall match the message with the oldest, incomplete SrcAvail advertisement. The Data Source shall view this SrcAvail as Processed and shall send the remaining ULP data using Data messages (see section A4.6.1 Bcopy on page 1228). The Data Sink shall consume this data before sending an RdmaRdCompl for other SrcAvail messages (this condition can only occur in Pipelined Mode - see section A4.8.3 Pipelined Mode on page 1251).

CA4-59: Upon receiving a SendSm message, the Data Source **shall** complete the oldest incomplete SrcAvail advertised buffer using Data message(s). After sending a SendSm message, the Data Sink **shall** wait until it has received all of the data that was advertised in the corresponding SrcAvail before sending any RdmaRdCompl message, even for another advertised buffer.

Note that the Data Sink calculates the number of remaining bytes expected through Data messages by subtracting from the SrcAH length field the sum of the number of bytes that the Data Sink has acknowledged with RdmaRdCompl message(s) plus the amount of ULP payload included in the SrcAvail message.

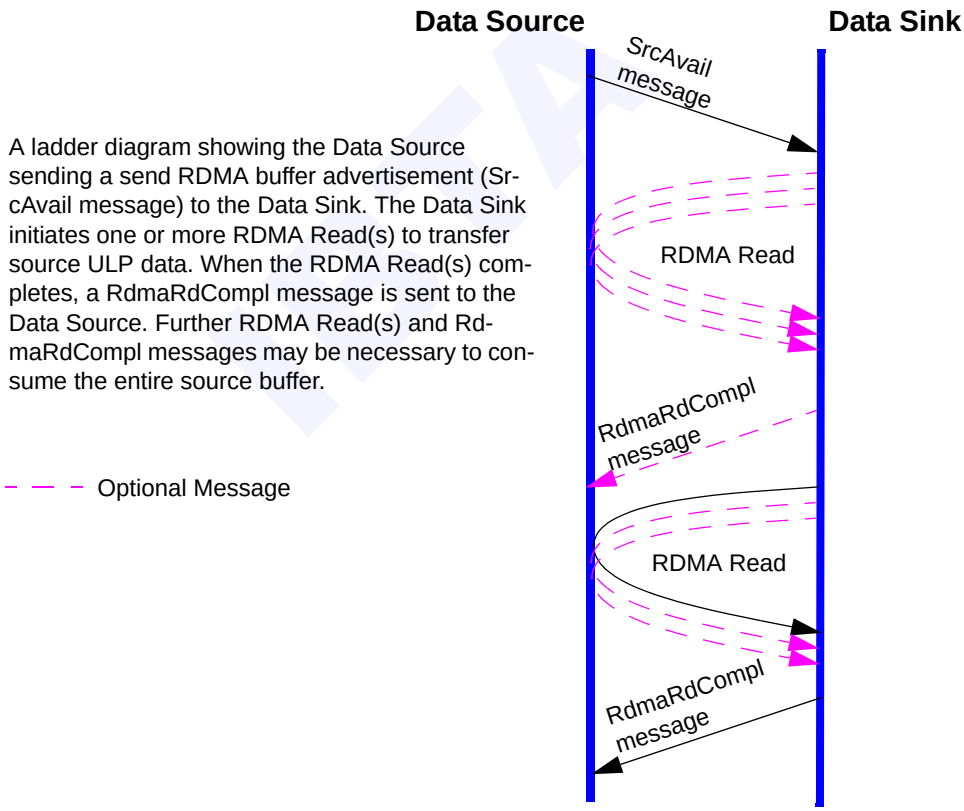


Figure 263 Ladder Diagram for Read Zcopy Mechanism

CA4-60: When the Data Sink sends multiple RdmaRdCompl messages for a single SrcAvail advertisement, the RdmaRdCompl message length field **shall** be the number of bytes transferred since the last RdmaRdCompl was sent for this SrcAvail advertisement.

It is possible to create a deadlock if, at the same time, both ULP peers post send data suitable for Read Zcopy and both ULPs wait for the associated send to complete before posting a receive. A SrcAvail message could be sent by each SDP peer, but no ULP receive buffer would be posted. This deadlock is possible when all of the following are true:

- A SrcAvail is received; and
- No ULP receive buffer is posted; and
- The local Data Source has a SrcAvail outstanding.

When these conditions are true, a deadlock can be avoided in a variety of ways:

- The Data Sink could send a SendSm message to force the use of the Bcopy data transfer mechanism.
- The Data Source could send a SrcAvailCancel message and then complete the ULP write using the Bcopy data transfer mechanism.
- The Data Sink could complete the Read Zcopy using a local buffer, holding that data until the ULP posts a receive.

Regardless of the method used, it is strongly recommended that SDP implementations detect and recover from this deadlock situation.

A4.6.3 WRITE ZCOPY

This mechanism shall transfer data through the following sequence of operations:

1) The Data Sink may send a SinkAvail message to the Data Source when a suitable receive ULP buffer is posted (note that the ULP buffer must be larger than the size of the local receive private buffers - see section A4.6.5.1 Detecting Stale SinkAvail Advertisements on page 1237). If Write Zcopy is chosen, the ULP buffer is referred to as a receive RDMA buffer (the receive RDMA buffer may actually be a private buffer from where receive data is copied to the ULP buffer -- this is implementation-dependent).

CA4-61: The SinkAH Len, VA, and R_Key fields **shall** describe the entire receive RDMA buffer. The NonDiscards field **shall** be set as specified in section A4.3.3.3.4 NonDiscards - 32 bits on page 1213.

CA4-61.2.1: In an *Invalidate Enabled* SDP connection, the R_Key in a *SinkAvail* message **shall be** capable of being remotely invalidated. Refer to the *Base Memory Management Extensions* for additional details on remote invalidation.

In an *Invalidate Enabled* SDP connection, the Data Sink should not rely on the Data Source to perform a remote invalidate after the data transfer is completed.

2) The Data Source receives the SinkAvail message and waits for the ULP to post a send buffer. If the Data Source determines the buffer is suitable for Write Zcopy, it shall use one or more RDMA Writes to transfer ULP data to the Data Sink. If the Data Source determines the buffer is unsuitable for Write Zcopy, it shall use the protocol described under section A4.6.5.2.2 Data Source Forcing Bcopy on page 1239

CA4-62: After the RDMA Writes(s) complete, the Data Source **shall** send a single RdmaWrCompl message to the Data Sink, unless the operation was canceled.

In an *Invalidate Enabled* SDP connection, the Data Source should send an RdmaWrCompl message, associated with a *SinkAvail* message, through a *Send w/Invalidate*; the R_Key used for the RDMA Write data transfer shall be used as an argument to the *Send w/Invalidate* verb.

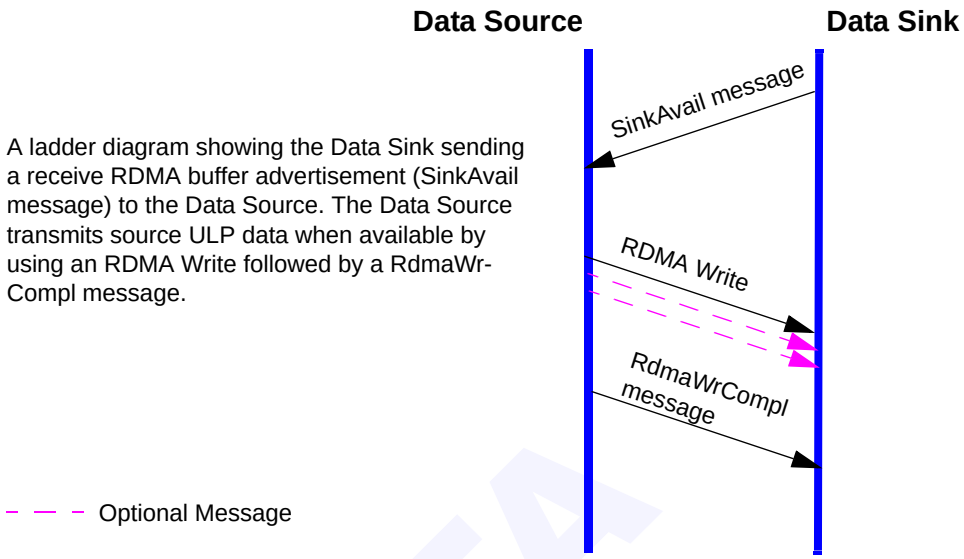
CA4-63: The RdmaWrCompl header **shall** contain the size (in bytes) of data transferred through the RDMA Write(s).

3) Upon receiving the RdmaWrCompl message, the Data Sink shall match the RdmaWrCompl message to the oldest incomplete SinkAvail advertisement and shall consider the SinkAvail advertisement Processed. Once the receive RDMA buffer is Processed, this may map to completion of a receive ULP buffer, as appropriate.

CA4-63.2.1: If the Data Source has remotely invalidated the RDMA buffer through a RdmaWrCompl message, the Data Sink shall verify that the invalidated R_Key is associated with the RDMA buffer sent in the oldest incomplete *SinkAvail* message. If there is a mismatch, the Data Sink shall view the operation as a protocol violation.

CA4-64: A Data Sink RDMA buffer advertised by SinkAvail **shall** remain available for Write Zcopy from the Data Source until it has been acknowledged with an RdmaWrCompl message, the SinkAvail was canceled due to a Data message (see section A4.6.5.1 Detecting Stale SinkAvail Advertisements on page 1237), or the advertisement has been revoked, (and

the revoke request has been Processed - see section A4.6.5.5 SinkAvail Revocation on page 1242).



A ladder diagram showing the Data Sink sending a receive RDMA buffer advertisement (SinkAvail message) to the Data Source. The Data Source transmits source ULP data when available by using an RDMA Write followed by a RdmaWr-Compl message.

Figure 264 Ladder Diagram for Write Zcopy Mechanism

Some socket implementations support an option to ensure that receive ULP buffers are completely filled before they are returned to the ULP. This is typically implemented as a flag called MSG_WAITALL which is specified when a receive ULP buffer is posted. If the MSG_WAITALL socket option is supported by the Data Sink implementation, it shall disable Write Zcopy for ULP buffers which have MSG_WAITALL set by not sending SinkAvail advertisements to the Data Source. Enabling Write Zcopy for buffers with MSG_WAITALL breaks the ULP buffer accounting algorithm which addresses crossing SinkAvail and Data messages (see section A4.6.5.1 Detecting Stale SinkAvail Advertisements on page 1237). If the ULP buffer accounting algorithm were used, then the ULP buffer must be partially completed if a Data message and SinkAvail message cross. Disabling SinkAvail prevents this condition, thus enabling the receive ULP buffer to be completely filled in all scenarios.

oA4-1: For any ULP buffer with the MSG_WAITALL socket option flag set, no SinkAvail advertisement **shall** be sent to the Data Source.

A4.6.4 TRANSACTION MECHANISM

If the ULP is transaction oriented, typically one peer is sending short *command* messages and medium to long *reply* messages are expected. It is possible to optimize this transfer model by collapsing the SinkAvail advertisement for the *reply* receive RDMA buffer with the Data message for the command. This enables zero-copy receives on potentially smaller *replies* as well as reducing control traffic. Note that the SinkAvail message is used

to transfer ULP payload which is being sent in the opposite direction of the SinkAvail message, and that for the SinkAvail to be generated the flow control mode must be Pipelined Mode (see section A4.8.3 Pipelined Mode on page 1251). Also note that the receive RDMA buffer for the SinkAvail advertisement must be larger than the local receive private buffer size (see section A4.6.5.1 Detecting Stale SinkAvail Advertisements on page 1237).

Note that the Data Sink can only send a SinkAvail message with ULP payload to the Data Source if the current flow control mode is Pipelined mode (see section A4.8.3 Pipelined Mode on page 1251).

If an end point receives a SinkAvail message with ULP payload, it may use RDMA Writes to fill the advertised RDMA buffer unless prior ULP payload-carrying messages effectively canceled this SinkAvail advertisement (see section A4.6.5.1 Detecting Stale SinkAvail Advertisements on page 1237).

Figure 265 Ladder Diagram of Transaction Mechanism on page 1236 shows the collapsing of the Data message into the SinkAvail advertisement. ULP peer A is communicating with ULP peer B, with a traffic pattern that appears as though it is transactional. Peer A is repetitively sending peer B a single small ULP message (this is the *command*) and then immediately posting a receive ULP buffer (this is the *reply*). Without the piggyback mechanism, two SDP messages would potentially be generated by peer A.

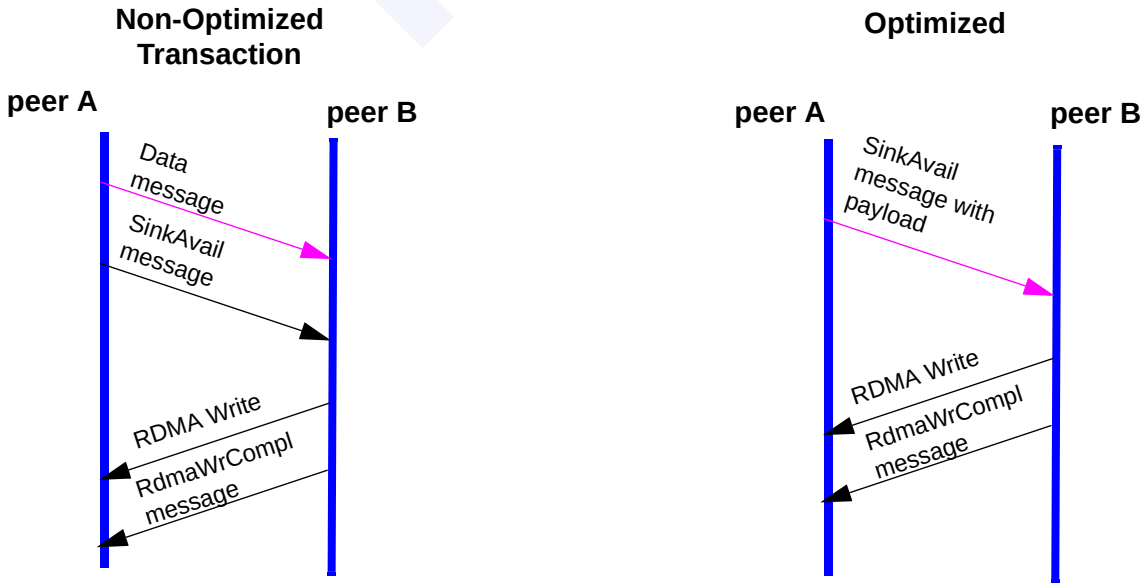


Figure 265 Ladder Diagram of Transaction Mechanism

ULP peer B is consistently waiting for reception of a *command* before posting a send ULP buffer for the *reply*. If the Transaction mechanism was not available and the SinkAvail advertisement was not received before the *reply* was posted by the ULP, the *reply* Data Source would have to choose whether to wait for the SinkAvail advertisement, generate a SrcAvail advertisement, or transfer the ULP data using the Bcopy mechanism. With the Transaction mechanism, the logic is straightforward. The *reply* Data Source may use the Write Zcopy mechanism to transfer the *reply*. If the *reply* ULP buffer is not suitable for RDMA Write, the Data Source may send the *reply* data using the Bcopy mechanism. All Pipelined Mode rules apply (see section A4.8.3 Pipelined Mode on page 1251).

A4.6.5 MISCELLANEOUS DATA TRANSFER ISSUES

A4.6.5.1 DETECTING STALE SINKAVAIL ADVERTISEMENTS

SDP allows the Data Source to send ULP data through SDP messages. This creates an issue in Pipelined Mode because a SinkAvail advertisement may cross an SDP message containing ULP data which is destined for the same receive ULP buffer that was advertised in the SinkAvail message. The receive ULP buffer could be at least partially satisfied through the Data message. This effectively requires the Data Source to view the receive RDMA buffer advertisement as outdated or stale.

CA4-65: The Data Sink **shall** only send a SinkAvail advertisement if the RDMA buffer is larger than the local receive private buffer size.

Note that this also means that the Data Sink Bcopy Threshold must be larger than or equal to the receive private buffer size.

SDP shall use the following algorithm to enable the Data Source to detect and recover from stale SinkAvail advertisements:

1) If a ULP receive RDMA buffer *R* has been advertised through a SinkAvail message and one or more messages with ULP payload (Data or SinkAvail message with ULP payload) arrive at the Data Sink, then the ULP payload of exactly one SDP message shall be copied to *R* and *R* shall be returned to the ULP. In other words, *R* will not consume the ULP payload of more than one SDP message.

CA4-66: In Pipelined mode, the ULP payload of only one SDP message (Data message or SinkAvail) **shall** be used to complete any one RDMA buffer advertised through SinkAvail.

If a receive ULP buffer has not been advertised through a SinkAvail message, it may consume the ULP payload of more than one SDP message.

2) The Data Source shall keep a 32-bit counter, *PotentialNonDiscards*, which tracks the number of SDP messages carrying ULP payload that the Data Source has sent that might not cause a SinkAvail message to be discarded. It is initialized to zero. It wraps to zero after reaching 0xFFFFFFFF.

3) The Data Source, upon sending an SDP message carrying ULP payload, shall increment *PotentialNonDiscards* by one.

4) At the Data Source:

CA4-67: The Data Source **shall** execute the following pseudo-code for each received SinkAvail advertisement before initiating a Write Zcopy data transfer using the advertised buffer. If ULP payload is present in the SinkAvail, it **shall** be processed normally regardless of whether the SinkAvail was discarded.

```
If (SinkAvail.NonDiscards!= PotentialNonDiscards)
  Discard(SinkAvail)
  PotentialNonDiscards--
Else
  Process SinkAvail normally
```

For example, if *PotentialNonDiscards*=2 and the Data Source has three SinkAvail advertisements, all with *NonDiscards*=0, then the first two advertisements are discarded as stale and RDMA is initiated on the third advertisement.

Note that this algorithm can cause receive ULP buffers to be partially filled when completed. If a ULP buffer is required to be completely filled an SDP implementation should not advertise the ULP buffer with a SinkAvail message. See section A4.6.3 Write Zcopy on page 1233 for additional details.

Detecting stale SinkAvail advertisements is one mechanism that causes a SinkAvail advertisement to be discarded. Section A4.9.4 Transition From Pipelined Mode to Combined Mode on page 1256 defines a different circumstance when a SinkAvail message must be discarded.

A4.6.5.2 MECHANISMS FOR FORCING BCOPY

A4.6.5.2.1 DATA SINK FORCING BCOPY

While in Combined or Pipelined Modes, if the Data Sink determines that its buffer is unsuitable for use with Read Zcopy from an RDMA buffer advertised by the Data Source, the Data Sink can use the SendSm message to force the Data Source to send data through the Bcopy mechanism (i.e., through Data messages). The Data Sink may send the SendSm message after receiving a SrcAvail message.

CA4-68: Upon receiving the SendSm message, the Data Source **shall** send all remaining ULP data (advertised in the associated SrcAvail message) using Data messages.

Section A4.7.1 SDP Message Ordering on page 1245 requires that the Data Sink process SrcAvail messages in MSeq order. For each SrcAvail message received, the Data Sink shall either proceed with the appropriate RDMA data transfer mechanism (Read Zcopy if a SinkAvail advertisement did not cross, or wait for an RdmaWrCompl if a crossing occurred - see section A4.8.3 Pipelined Mode on page 1251) or it shall respond with a SendSm message.

The Data Source shall respond to the SendSm request by matching it to the oldest incomplete SrcAvail advertisement and then sending the remaining ULP data for the SrcAvail advertisement (i.e. that has not been Processed by RdmaRdCompl message(s) from the Data Sink or already sent as ULP payload in the SrcAvail message) through the Bcopy mechanism.

Implementation note: in some cases the Data Source and Data Sink will have different Bcopy Threshold values. When the Source advertises an RDMA buffer whose size is greater than the Source's Bcopy Threshold but less than the Sink's Bcopy Threshold, the Sink may choose to force a Bcopy. However, since the Source has already invested in setting up a Read Zcopy data transfer, the Sink should give special consideration to cooperating with the Source's attempt to use Read Zcopy. Still, the Sink is free to force a Bcopy if it determines that for any reason its buffer is unsuitable for Read Zcopy.

A4.6.5.2.2 DATA SOURCE FORCING BCOPY

While in Pipelined Mode, if the Data Source determines that its buffer is unsuitable for use with Write Zcopy to an RDMA buffer advertised by the Data Sink, the Data Source is allowed to choose to not use the Data Sink's RDMA buffer advertisement, and use Data messages to send data using the Bcopy data transfer mechanism. In this case, the Data Source and Data Sink shall follow the protocol described in section A4.6.5.1 Detecting Stale SinkAvail Advertisements on page 1237.

Implementation note: in some cases the Data Source and Data Sink will have different Bcopy Threshold values. When the Data Sink advertises an RDMA buffer whose size is greater than the Data Sink's Bcopy Threshold but less than the Data Source's Bcopy Threshold, the Data Source may choose to force a Bcopy. However, since the Data Sink has already invested in setting up a Write Zcopy data transfer, the Data Source should give special consideration to cooperating with the Data Sink's attempt to use Write Zcopy. Still, the Data Source is free to force a Bcopy if it determines that for any reason its buffer is unsuitable for Write Zcopy.

A4.6.5.3 PROCESSING OUT-OF-BAND DATA

CA4-69: When the Data Source ULP posts Out-Of-Band data (a single byte) to be transmitted, the ordering of the Out-Of-Band data in the output byte stream **shall** be preserved.

The precise mechanism for conveying OOB data requests from the ULP to the SDP implementation is outside the scope of this specification.

CA4-70: Once the ULP has indicated that a particular byte in its output stream should be marked as Out-Of-Band data, the SDP implementation **shall** notify the remote peer that out-of-band data is pending by setting the OOB_PEND flag on an outgoing SDP message. It is recommended that this notification be accomplished in an expeditious fashion, however, the only requirements levied by the specification are as follows:

- The OOB_PEND flag **shall** be sent exactly once for each OOB data indication.
- The OOB_PEND flag **shall** be sent on a message sent no later than the message containing the Out-Of-Band data byte.
- The implementation **shall**, if necessary, delay sending the OOB_PEND flag to ensure that there will be no more than 65,535 (2^{16}-1) bytes of data sent between the flag and its associated Out-Of-Band data byte. This includes all ULP data sent by any SDP data transfer mechanism, including any data sent in or advertised by the SDP message containing the OOB_PEND flag.

Note that an implementation is allowed to send a message with the OOB_PEND flag using a reserved credit. See section A4.7.5 Use of Send Credits on page 1246.

CA4-71: When the output byte-stream advances to the point where the Out-Of-Band data was inserted into the data stream by the ULP, the Data Source **shall** send the Out-Of-Band data using a Data message with the OOB_PRES bit set and the Out-of-Band data byte as the last byte of the ULP payload in the Data message.

Upon receipt of an SDP message with the OOB_PEND flag set, it is recommended that the SDP implementation expeditiously notify the ULP that OOB data is pending; however, the precise mechanism for conveying OOB notifications from the SDP implementation to the ULP is outside the scope of this specification.

A4.6.5.4 SRCAVAIL REVOCATION

To revoke all incomplete SrcAvail messages sent by the Data Source to the Data Sink, the Data Source shall send a SrcAvailCancel message. This is needed, for example, if the ULP performs a socket write and a timeout capability is supported. If the timeout interval passes without suc-

cessful completion of the transfer all RDMA buffers advertised on behalf of the socket write need to be canceled. Rather than create a new message type to explicitly acknowledge the SrcAvailCancel message, the SendSm message is used because it can be unambiguously understood to complete the cancel operation.

The Data Sink, upon receiving the SrcAvailCancel message, shall invalidate all Unprocessed SrcAvail messages (SrcAvail messages which have not been operated on), and should invalidate all In-Process SrcAvail messages (RDMA Read processing has started, but an RdmaRdCompl or SendSm message to complete the SrcAvail advertisement has not been sent) -- see details below. If all SrcAvail messages have been Processed, then the SrcAvailCancel message shall be ignored.

If a SrcAvail message is In-Process at the Data Sink (i.e., it has initiated one or more RDMA Reads), the RDMA Read cannot be canceled.

CA4-72: The Data Sink **shall not** update the value of MSeqAck to be sent in SDP messages, to greater than or equal to the MSeq value, with wrap, in the SrcAvailCancel message until all In-Process SrcAvail advertisements have been completed with the following sequence of events:

1) The Data Sink **shall not** initiate any new RDMA Reads.
2) After completion of all In-Process RDMA Reads, the Data Sink **shall** send any relevant RdmaRdCompl messages (this may or may not complete the SrcAvail message, depending on how many bytes have been consumed).
3) If there is more ULP data which has not been transferred from the original SrcAvail message, the Data Sink **shall** cancel the remainder of the SrcAvail advertisement.

CA4-73: If the Data Sink canceled one or more SrcAvail advertisements (either Unprocessed or In-Process), the Data Sink **shall** send exactly one SendSm message associated with the oldest incomplete SrcAvail message.

CA4-74: The Data Source, after sending a SrcAvailCancel message, **shall not** send any new SrcAvail messages until the SrcAvailCancel message has been Processed, as defined below.

This enables the Data Sink to implement simpler accounting (i.e., not have to account for whether a SrcAvail message was sent before or after the SrcAvailCancel message).

The Data Source shall consider the SrcAvailCancel message Processed if any of the following occur:

- All Unprocessed or In-Process SrcAvail messages have been moved to the Processed state with an RdmaRdCompl message or have been over-ridden by a SinkAvail message (see section A4.8.3 Pipelined Mode on page 1251).
- A SendSm message is received with an MSeqAck value greater than or equal to the MSeq value in the SrcAvailCancel message.

A4.6.5.5 SINKAVAIL REVOCATION

To revoke all incomplete SinkAvail advertisements sent by the Data Sink to the Data Source, the Data Sink shall send the SinkAvailCancel message. This is needed, for example, if the ULP performs a socket read and a timeout capability is supported. If the timeout interval passes without successful completion of the transfer all RDMA buffers advertised on behalf of the socket read need to be canceled.

The Data Source, upon receiving the SinkAvailCancel message, shall invalidate all Unprocessed SinkAvail advertisements (SinkAvail messages which have not been operated on) and should cancel all In-Process SinkAvail messages (RDMA Write processing has started, but an RdmaWrCompl message which completes the SinkAvail advertisement has not been sent) -- see details below. If all SinkAvail advertisements have been Processed, the SinkAvailCancel message shall be ignored.

Because an RDMA Write can not be canceled, if a SinkAvail message is In-Process at the Data Source (i.e., it has initiated one or more RDMA Writes), the RDMA Write shall be allowed to complete and the RdmaWrCompl message shall be sent.

CA4-75: The Data Source **shall** complete the buffer with the following sequence of events:

1) The Data Source **shall not** initiate any new RDMA Writes
2) After completion of the In-Process RDMA Writes, the Data Source **shall** send any relevant RdmaWrCompl messages (this may or may not complete the SinkAvail message, depending on how many bytes have been consumed)
3) if there is more ULP data which has not been transferred into the RDMA buffer advertised by the SinkAvail message, the Data Sink **shall** invalidate the remainder of the SinkAvail advertisement.

CA4-76: If the Data Source canceled one or more SinkAvail advertisements (either Unprocessed or In-Process), the Data Source **shall** send exactly one SinkCancelAck message.

CA4-77: The Data Sink, after sending the SinkAvailCancel message, **shall not** send a new SinkAvail or SinkAvailCancel message until all previous SinkAvail messages have been Processed, as defined below.

This enables the Data Source to implement simpler accounting (i.e., not have to account for whether a SinkAvail message was sent before or after the SinkAvailCancel message).

The Data Sink shall consider the SinkAvailCancel message Processed if any of the following occur:

- All Unprocessed or In-Process SinkAvail messages have been moved to the Processed state with an RdmaWrCompl message (i.e. the byte count returned in the RdmaWrCompl completely consumed the buffer).
- a Data message that adhered to the stale advertisement rules (see section A4.6.5.1 Detecting Stale SinkAvail Advertisements on page 1237).
- A SinkCancelAck message is received.

A4.6.5.6 BUFFERING ULP PAYLOAD

Under certain conditions it is possible for a sockets application to deadlock unless ULP payload is buffered by the underlying sockets implementation. For example, if all of the following occur:

a) both ULP peers perform a sockets send followed by a sockets receive,

b) both ULP peers do not post the receive buffer until the send is completed, and

c) the underlying sockets implementation does not buffer the send data

then the send will never complete - thus creating deadlock. To solve this in the most general case (i.e. infinite length sends) is intractable, thus existing sockets applications bound the amount of buffering required by the transport layer through the use of the socket options SO_RCVBUF and SO_SNDBUF.

An application whose behavior is similar to the above example will not deadlock if the application ensures that a send is never larger than the size of the local peer's SO_RCVBUF plus the remote peer's SO_SNDBUF, and the SDP implementation ensures there is SO_RCVBUF plus SO_SNDBUF amount of buffering in the local and remote peer respectively.

Note that an application may post buffers larger than SO_RCVBUF plus SO_SNDBUF - but to remain deadlock free it must ensure that it does not exhibit the above behavior (e.g. a backup application could post large sends in one direction after it is sure the remote peer is posting receives).

Specification of the exact buffering algorithm is beyond the scope of this specification, but care must be taken if the receive private buffer pool is used as part of the SO_RCVBUF buffers. This is because an entire receive private buffer may, in some situations, contain only one byte of ULP data instead of being filled completely.

Thus an SDP implementation should provide at least SO_RCVBUF amount of buffering for ULP data at the Data Sink. An SDP implementation should provide at least SO_SNDBUF amount of ULP data buffering at the Data Source.

A4.7 PRIVATE BUFFER MANAGEMENT

SDP uses credit-based flow control on a per-socket connection basis. Each peer – for each connection – posts some number of private buffers as receive requests to the Receive Queue of the QP associated with the socket. The number of currently posted receive private buffers is advertised by the local peer to the remote peer in the Bufs field in the BSDH of each SDP message.

CA4-78: Private buffers **shall** obey the following enumerated constraints:

a) All receive private buffers **shall** be at least as large as the advertised buffer size. See section A4.7.6 Receive Buffer Resizing on page 1247 for receive private buffer constraints when resizing.

b) The total number of receive private buffers **shall** be at least 3 per-connection for normal data flow. It is recommended the number of receive private buffers be substantially greater than 3.

c) A local peer **shall not** send SDP messages larger than the size of the remote peer's receive private buffers.

d) The sizes of both send and receive private buffers **shall** be at least the size of the BSDH plus the size of the largest extended header in an SDP message (which is SinkAH) plus one byte.

e) The Data Sink Bcopy Threshold **shall** be greater than or equal to the size of the local peer's receive private buffers.

f) the number of posted private buffers shall not exceed HCA or QP limits.

Receive private buffers should be substantially larger than the minimum value to enable practical data transfer using the Bcopy mechanism.

In addition, send buffers may obey the following constraint:

- If the local peer's send buffer size is larger than the remote peer's receive private buffer size, the local peer may reduce the size of its send buffers or leave them unmodified. The latter approach may be advantageous if the remote peer enlarges its receive private buffers at a later time.

A4.7.1 SDP MESSAGE ORDERING

CA4-79: The SDP sender **shall** insert messages into the Send Queue in BSDH MSeq order.

Note that this means the SDP message MSeq value in the BSDH will be monotonically increasing in the Send Queue.

CA4-80: The SDP receiver **shall** process all messages in BSDH MSeq order.

A4.7.2 SEND CREDIT CALCULATION

Send credit is calculated using information in the BSDH included with each SDP message.

Consider the case of peer 1 sending a SDP message to its connected peer, peer 2. The header of the SDP message includes the number of receive private buffers peer 1 currently has posted on that connection (in the Bufs field of the BSDH). The header also includes the sequence number of the last SDP message peer 1 has received before sending this SDP message (in the MSeqAck field of the BSDH - see section A4.6.5.4 SrcAvail Revocation on page 1240 for additional constraints on MSeqAck). Upon receiving this SDP message, peer 2 uses this information to update its send credit for that connection:

New send credit = bufs – WrapSubtract(*LSSeq* – MSeqAck*)*

where *LSSeq* ("Last Sent Sequence number") is the MSeq of the last SDP message sent by peer 2.

See section A4.7.5 below for the detailed rules governing usage of available send credits.

A4.7.3 INITIALIZATION OF SEND CREDIT

Initial send credit advertisements are exchanged during connection setup in the Buf field of the BSDH within the Hello and HelloAck messages. These credit advertisements shall be greater than or equal to 3. Either before or after connection setup, the receiver may post additional receive private buffers and increase the advertised window.

CA4-81: On connection establishment, initial credit advertisement from each peer **shall** be no less than three.

A4.7.4 GRATUITOUS UPDATE OF THE REMOTE PEER'S SEND CREDIT

As previously mentioned, credit updates are included in the header of each SDP message. Therefore when ULP data flow is such that SDP message flow is bidirectional (e.g., when doing Zcopy data transfer), credits are refreshed as part of the data transfer process. In some scenarios, bidirectional SDP message flow does not occur. Under these circumstances, SDP shall send gratuitous Data messages (Data messages with no ULP payload) as required to update the remote peer's send credit.

A4.7.5 USE OF SEND CREDITS

The sender shall reserve two receive private buffer credits to ensure the SDP connection operates correctly under flow controlled conditions.

The sender must reserve one credit for an SDP message which provides additional credits. If this credit was not reserved, a deadlock scenario is possible if both ends become flow controlled. Reserving a receive private buffer for the flow control update ensures that the sender can always update the receiver when more receive private buffers are posted.

The sender must reserve one additional credit for sending any SDP message which does not contain ULP payload. This ensures that the credit can be refreshed by the remote peer without depending upon ULP receive behavior. If ULP payload was allowed to be present in the SDP message, it is possible to have protocol deadlock.

CA4-82: Before sending any SDP message over the IB RC connection, an SDP implementation **shall** compute its available send credit as detailed in section A4.7.2, and **shall** then obey the following rules:

- If no credits are available, an implementation **shall not** send any type of SDP message.
- If one credit is available, an implementation **shall** only send SDP messages that provide additional credits and also do not contain ULP payload.
- If two credits are available, an implementation **shall** only send SDP message which do not contain ULP payload.
- An SDP implementation **shall** send an SDP message that provides additional credit(s) if the remote side's credits drop to one or fewer credits. The sending of this message by the local side **shall not** be contingent upon the local side first receiving some other SDP message from the remote side.

Note that if three or more credits are available, an implementation may send any type of SDP message which would otherwise be legal.

A4.7.6 RECEIVE BUFFER RESIZING

The local peer may request the remote peer to change its receive private buffer pool buffer size by sending a Change Receive Buffer message (ChRcvBuf) with the desired new size. This enables the local peer to increase or decrease the maximum size of its outgoing messages if the remote peer agrees to the change. See section A4.7 Private Buffer Management on page 1244 for restrictions on the size of the receive private buffers.

The remote peer should change the size of its receive private buffers to the desired size specified in the ChRcvBuf message; it may make them larger than the desired size, for example for alignment or performance optimization. If the remote peer is unable or unwilling to change its receive private buffer size in this manner, it should change the size to be as close as possible.

CA4-83: Upon receipt of ChRcvBuf message, the remote peer **shall not** change the buffer size in the direction opposite of that requested.

That is, if the local peer requests a decrease, the remote peer shall either decrease the size or leave it unchanged. Conversely, if the local peer requests an increase, the remote peer shall either increase the size or leave it unchanged.

CA4-84: Upon receipt of the ChRcvBuf message, if the local peer requested a decrease in the private buffer size, the remote peer **shall not** decrease the private buffer size to be smaller than that requested.

CA4-85: To confirm the change, the remote peer **shall** send a ChRcvBufAck message with the new size of its receive private buffers when all previous receive private buffers of smaller size have been consumed.

The ChRcvBufAck message may be sent immediately if the new size is smaller than or equal to the old size. If the new size is larger than the old size, the ChRcvBufAck message shall be sent after all receive private buffers of the old size have been consumed. If the remote peer is unable to resize its receive private buffers, it shall specify in the ChRcvBufAck message the original receive private buffer size.

The remote peer shall continue to use the old receive private buffer size to determine whether a SinkAvail message can be sent for a specific ULP buffer until it has sent the ChRcvBufAck message. At that time the remote peer shall use the new receive private buffer size to determine whether a SinkAvail message may be sent.

After receiving the ChRcvBufAck message, the local peer may change the maximum size of an SDP send message to the value specified in the ChRcvBufAck message.

CA4-86: The local peer **shall not** increase the maximum size of an SDP send message until the ChRcvBufAck message is received.

CA4-87: If the ChRcvBuf message requested an increased size and the ChRcvBufAck message contains a size that is the same as the original size before the ChRcvBuf message was sent, then the local peer **shall not** request any further size increases for this connection.

CA4-88: If the ChRcvBuf message requested a decreased size and the ChRcvBufAck message contains a size that is the same as the original size before the ChRcvBuf message was sent, then the local peer **shall not** request any further size decreases for this connection.

CA4-89: The local peer **shall not** send a new ChRcvBuf message if there is an unacknowledged ChRcvBuf message.

A4.7.6.1 CONFLICT RESOLUTION

CA4-90: If both peers concurrently send each other ChRcvBuf messages, then the accepting peer **shall** disregard the ChRcvBuf message.

The connecting peer shall respond to the ChRcvBuf message. The connecting peer may re-send its ChRcvBuf message after sending the ChRcvBufAck message in response to the accepting peer's ChRcvBuf message.

A4.7.6.2 FLOW CONTROL ISSUES DURING RESIZING

When a peer receives the ChRcvBuf message and it decides to change its receive private buffer size in response to this request, the peer must allocate new receive private buffers of the desired size and post these private buffers to the Receive Queue. The peer shall not wait for completion of all posted receive private buffers of the previous size before allocating and posting the new (different size) receive private buffers. This is required to enable the remote peer to continue sending messages which will cause the old-size receive private buffers to complete; otherwise the remote peer will stop sending messages once the channel becomes stalled and the reserved receive private buffers will not be consumed.

A4.8 SDP MODES

SDP modes control how ULP buffers larger than the Bcopy Threshold are transferred. Data Source ULP buffers less than or equal to the Bcopy Threshold are sent using the Bcopy or Transaction mechanism (note that because the Bcopy Threshold is locally defined, it may be fixed, variable, or be defined as infinite - which would cause the Data Source to always

use the Bcopy mechanism). ULP buffers larger than the Bcopy Threshold are sent in a variety of ways depending upon the current mode. The three modes are:

- Combined Mode - the initial mode. This mode enables both Bcopy and Read Zcopy data transfer mechanisms, but with only one outstanding Read Zcopy operation at a time. The SrcAvail message contains a non-zero length ULP payload.
- Pipelined Mode - All data transfer mechanisms are valid, including multiple outstanding transfers at one time (with some limits). The SrcAvail message contains no ULP payload.
- Buffered Mode - only the Bcopy data transfer mechanism is valid. The main difference between this mode and Combined Mode is that the Data Source cannot generate SrcAvail messages.

In all modes the Data Sink may force the Data Source to transfer data via the Bcopy mechanism. In Buffered Mode the Data Source always uses the Bcopy mechanism. In Combined or Pipelined mode, the Data Sink may force data transfer using the Bcopy mechanism by issuing a SendSm message. In this case, however, an extra round trip is required to cause the Bcopy mechanism to be used because of the SrcAvail/SendSm sequence. Buffered Mode eliminates this extra overhead.

Pipelined Mode is the highest performance mode. It enables multiple outstanding zero-copy transfers, optimizing for either the Data Sink ULP buffer being posted first (Write Zcopy) or the Data Source ULP buffer being posted first (Read Zcopy). Pipelined Mode also enables mixing of Bcopy and Zcopy mechanisms.

CA4-91: The data flow mode between peers **shall** be independent in each direction.

For example, data flow from the local peer to its remote peer may be in Buffered mode in one direction, but the reverse direction may be in Combined mode. Table 356 summarizes the various characteristics of each mode. Table 357 summarizes the possible actions at the local peer A for each combination of modes between the local peer A and remote peer B. In the table, a “1” in the Send or Accept column indicates that the specific message type is valid, but only one can be outstanding at a time.

Table 356 Mode Characteristics

Mode	Multiple Outstanding Zcopy Requests	Simultaneous Outstanding SinkAvail & SrcAvail	Data in SrcAvail	Mix ULP Data Messages with Write Zcopy	Mix ULP Data Messages with Read Zcopy	List of Available Transfer Mechanisms
Buffered	N/A	N/A	N/A	N/A (Write Zcopy not allowed)	N/A (Read Zcopy not allowed)	Bcopy, Transaction*
Combined	No	No	Yes	N/A (Write Zcopy not allowed)	Yes, but not at same time	Bcopy, Read Zcopy, Transaction*
Pipelined	Yes	Yes	No	Yes	Yes, but not at same time	Bcopy, Read Zcopy, Write Zcopy, Transaction*

*if the reverse half-connection is in Pipelined Mode

Table 357 Summary of Permitted Actions By Mode Pair

Half Connection		Host A is Allowed To:							
A to B	B to A	Post		Send			Accept		
		RDMA Read Req	RDMA Write	SrcAvail	SinkAvail	Trans-action	SrcAvail	SinkAvail	Trans-action
Buffered	Buffered	NO	NO	NO	NO	NO	NO	NO	NO
Buffered	Combined	YES	NO	NO	NO	NO	1	NO	NO
Buffered	Pipelined	YES	NO	NO	YES	YES	YES	NO	NO
Combined	Buffered	NO	NO	1	NO	NO	NO	NO	NO
Combined	Combined	YES	NO	1	NO	NO	1	NO	NO
Combined	Pipelined	YES	NO	1	YES	YES	YES	NO	NO
Pipelined	Buffered	NO	YES	YES	NO	NO	NO	YES	YES
Pipelined	Combined	YES	YES	YES	NO	NO	1	YES	YES
Pipelined	Pipelined	YES	YES	YES	YES	YES	YES	YES	YES

A4.8.1 BUFFERED MODE

CA4-92: In Buffered Mode all data **shall** be transferred using the Bcopy mechanism or optionally, if the opposite half-connection is in Pipelined Mode, the Transaction mechanism.

Thus only Data or SinkAvail messages may used to transfer ULP data.

A4.8.2 COMBINED MODE

In Combined Mode, if the send ULP buffer is less than or equal to the Data Source Bcopy Threshold, the Data Source uses the Bcopy mechanism (i.e., by sending Data messages) or optionally, if the opposite half connection is in Pipelined Mode, the Transaction mechanism. If the ULP buffer is larger than the Bcopy Threshold, data is transferred using the Read Zcopy mechanism.

CA4-93: In Combined Mode the Data Sink **shall** be prepared to receive ULP data through either the Bcopy mechanism, the Read Zcopy mechanism, or the Transaction mechanism.

CA4-94: In Combined Mode, if the Read Zcopy mechanism is used, after the Data Source sends a SrcAvail message it **shall not** send any messages which contain a ULP payload until all data transfer associated with the SrcAvail message is complete (specifically, a RdmaRdCompl or SendSm message is received).

This effectively means that only a single SrcAvail message may be In-Process at any one time, and the Data Source can not use the Bcopy data transfer mechanism if a Read Zcopy is In-Process.

CA4-95: In Combined Mode, the SrcAvail message **shall** contain greater than zero bytes of ULP payload. The actual amount of ULP data included is implementation dependent.

A4.8.3 PIPELINED MODE

In Pipelined Mode, if the ULP buffer is less than or equal to the Bcopy Threshold, the Data Source uses the Bcopy mechanism (e.g., by sending Data messages) or optionally, if the opposite half connection is in Pipelined Mode, the Transaction mechanism. If the ULP buffer is larger than the Bcopy Threshold, data is transferred using the Read Zcopy mechanism or the Write Zcopy mechanism.

CA4-96: In Pipelined Mode the Data Sink **shall** be prepared to receive ULP data through any of the data transfer mechanisms.

CA4-97: In Pipelined Mode, the Data Source **shall not** include ULP payload in any SrcAvail messages.

CA4-98: After sending one or more SrcAvail messages, the Data Source **shall not** send any messages with a ULP payload until all data transfers associated with previously sent SrcAvail message(s) have been Processed. The single exception to this is the case when the Data Sink sends a SendSm message. In this case, the Data Source **shall** send the remaining data associated with the SrcAvail through Data messages. The remaining Unprocessed or In-Process SrcAvail advertisements remain valid and **shall** be Processed by the Data Sink after it consumes these Data messages.

This restriction is necessary since a crossing SinkAvail message would cancel any advertised SrcAvails, and the Sink would be unable to process the received in-line ULP payload until it received and processed all the data associated with the canceled SrcAvails. See A4.6.2 Read Zcopy on page 1229 for further details.

The Data Sink may also advertise receive RDMA buffers using SinkAvail messages (i.e., Write Zcopy mechanism, which uses RDMA Writes). If SrcAvail and SinkAvail messages cross, then Write Zcopy has higher priority than Read Zcopy (i.e., the SrcAvail messages are canceled).

CA4-99: The following rules for crossing SinkAvail/SrcAvail advertisements **shall** be obeyed. If both compliance statements 4-67 (4-67 describes discarding stale SinkAvail messages) and 4-99 apply, then compliance statement 4-67 **shall** take precedence over 4-99.

a) If the Data Source receives a SinkAvail message:

   i) the Data Source **shall** use the Write Zcopy mechanism to transfer data - even if it has already advertised the ULP send data through a SrcAvail message.

   ii) the Data Source **shall** treat all outstanding SrcAvail advertisements as having been discarded by the Data Sink.

      - This implies that if the Data Source consumes all SinkAvail advertisements and ULP send data remains that is suitable for RDMA, the Data Source should advertise the ULP data through a SrcAvail message, even if a SrcAvail advertisement for that send RDMA buffer was sent prior to receiving the SinkAvail message.

b) If the Data Sink receives a SrcAvail message

   i) and it has no Unprocessed or In-Process SinkAvail message, the Data Sink **shall not** send a SinkAvail message and the Data Sink **shall** transfer data using the Read Zcopy mechanism.

ii) and a SinkAvail message is Unprocessed or In-Process, the Data Sink **shall** discard all Unprocessed or In-Process SrcAvail advertisements and **shall** ignore the current SrcAvail advertisement (but otherwise process the packet normally, e.g. flow control information, etc.).

In Pipelined mode multiple send/receive RDMA buffers may be advertised by sending multiple SrcAvail/SinkAvail messages without waiting for data transfers associated with prior SrcAvail/SinkAvail messages to complete.

CA4-100: The Data Sink **shall** limit the maximum number of outstanding SrcAvail and SinkAvail advertisements to the HH or HAH MaxAdverts value specified by the remote peer during connection setup.

A4.9 SDP MODE TRANSITIONS

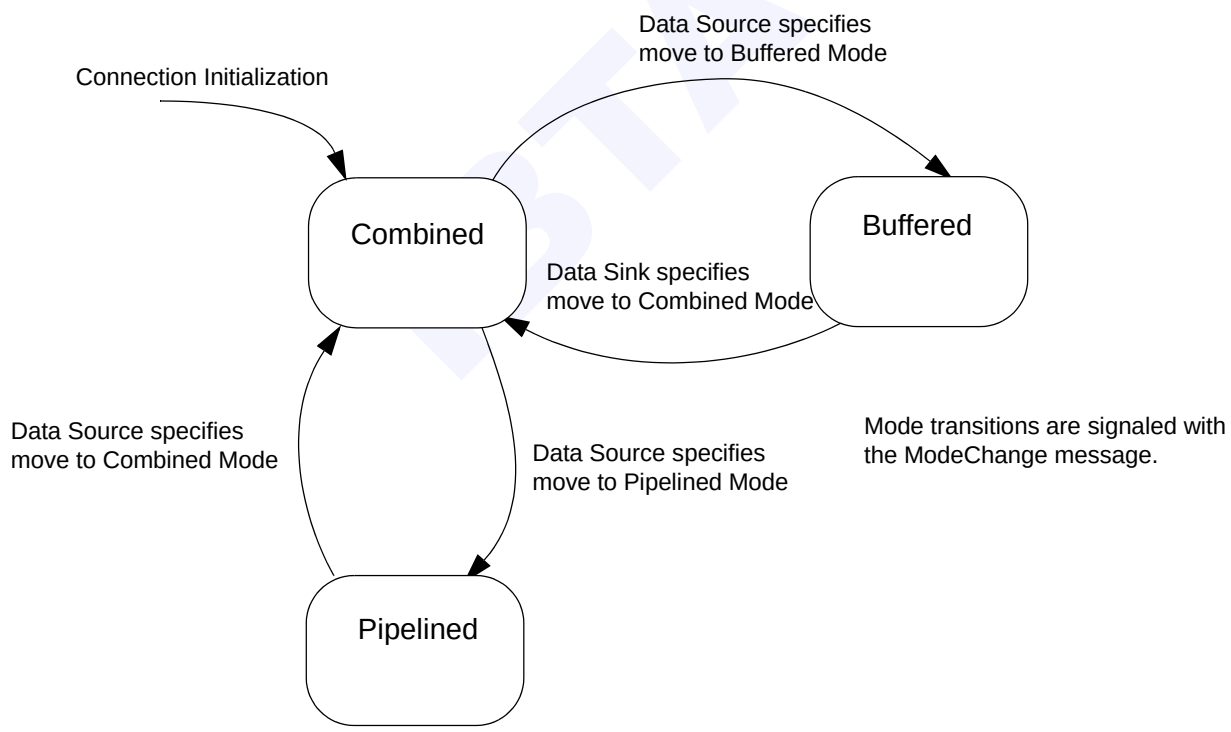


Figure 266 Mode State Machine

CA4-101: Figure 266 defines the possible transitions between the SDP modes. Each SDP mode **shall** have a master, which controls any mode changes, and a slave, which passively changes mode when told by the master, as specified in Table 358.

CA4-102: An SDP implementation **shall not** send a ModeChange message which specifies the current mode.

The slave may indicate to the master that a mode change is recommended by either setting the REQ_PIPE flag in the BSDH (transition from Combined Mode to Pipelined Mode) or by using SendSm to transfer ULP data (transition from Pipelined Mode to Combined Mode or from Combined Mode to Buffered Mode). The master may choose to ignore the request.

Table 358 Mode Master

Mode	Master	Mode Change Hint From Slave
Buffered	Data Sink	None
Combined	Data Source	SendSm messages or REQ_PIPE
Pipelined	Data Source	SendSm messages

CA4-103: The master **shall** change its mode immediately after sending a ModeChange message.

This may occur before the completion event associated with completion of the Reliable Connection transfer of the ModeChange message. In a specific mode, the master shall only use data transfer mechanisms allowed for that mode (see section A4.8 SDP Modes on page 1248). For example, if the prior mode was Combined Mode and the current mode is Buffered Mode, the Data Source shall not generate SrcAvail messages.

CA4-104: When the slave receives the ModeChange message, the slave **shall** immediately set its current mode to the mode specified in the ModeChange message.

In a specific mode, the slave shall only use data transfer mechanisms allowed for that mode. For example, if the prior mode was Pipelined Mode and the current mode is Combined Mode, the Data Sink (slave) must not generate SinkAvail messages.

Depending on the mode transition, the master and slave may be required to take some further actions, as described later in this chapter.

Note that when a connection is first set up, the local peer shall set the initial mode for the local Data Sink and Data Source to be Combined Mode (see section A4.5.1 Connection Setup on page 1218).

The Data Source (master) may cause a transition to either Buffered Mode or Pipelined Mode.

When in Buffered Mode or Pipelined Mode, the only legal mode transition is to Combined Mode.

Following subsections give details of SDP message exchanges needed to transition from one mode to another. Each transition is caused by sending a ModeChange message.

A4.9.1 TRANSITION FROM COMBINED MODE TO BUFFERED MODE

CA4-105: To transition from Combined to Buffered mode, the Data Source (master) **shall** send a ModeChange message with the MCH fields set as follows:

- S=0, (i.e., change the Data Sink mode)
- Mode = BUFF_MODE (see section A4.3.3.8.2 Mode - 3 bits on page 1215)

CA4-106: The Data Source **shall not** send the ModeChange message until all Unprocessed or In-Process Read Zcopy transfers have been moved to the Processed state with either a RdmaRdCompl or SendSm message from the Data Sink (there will be at most one outstanding in Combined Mode).

This ensures that no messages specific to Combined Mode (i.e., those related to the Read Zcopy mechanism) can be received by either peer when in Buffered Mode.

A4.9.2 TRANSITION FROM BUFFERED MODE TO COMBINED MODE

CA4-107: To transition from Buffered to Combined mode, the Data Sink (master) **shall** send a ModeChange message with the MCH fields set as follows:

- S=1 (i.e., change the Data Source mode)
- Mode = COMB_MODE (see section A4.3.3.8.2 Mode - 3 bits on page 1215)

Because all SDP message types which are legal for Buffered Mode are also legal for Combined Mode, no special action is needed for this transition.

A4.9.3 TRANSITION FROM COMBINED MODE TO PIPELINED MODE

CA4-108: To transition from Combined to Pipelined mode, the Data Source (master) **shall** send a ModeChange message with the MCH fields set as follows:

- S=0 (i.e., change the Data Sink mode)
- Mode = PIPE_MODE (see section A4.3.3.8.2 Mode - 3 bits on page 1215)

Because all SDP message types which are legal for Combined Mode are also legal for Pipelined Mode, no special action is needed for this transition. For example, if the Data Source has an outstanding SrcAvail advertisement (and there can be at most one such advertisement outstanding in Combined Mode), then it need not wait for a RdmaRdCompl or SendSm before sending the ModeChange message.

A4.9.4 TRANSITION FROM PIPELINED MODE TO COMBINED MODE

CA4-109: To transition from Pipelined to Combined mode, the Data Source (master) **shall** send a ModeChange message with the MCH fields set as follows:

- S=0 (i.e., change the Data Sink mode)
- Mode = COMB_MODE (see section A4.3.3.8.2 Mode - 3 bits on page 1215)

CA4-110: After sending the ModeChange message, the Data Source immediately transitions to Combined Mode, but Data Source **shall not** subsequently switch to any other mode until it receives an SDP message with the following constraint:

MSeqAck >= (MSeq of the ModeChange message)

In the above calculation, MSeqAck and “MSeq of the ModeChange message” are treated as signed integers.

This constraint ensures that all stale SinkAvail messages will be received at the Data Source before a transition out of Combined Mode. The behavior upon receiving a stale SinkAvail message while in Combined Mode is described later in this section.

If data transfer is occurring, the Data Source is guaranteed to eventually receive the acknowledgement for the ModeChange message. If no data transfer is occurring, the acknowledgement can take an indeterminate amount of time. However there is no need to immediately switch out of Combined Mode if no data transfer is occurring.

The transition from Pipelined to Combined Mode imposes additional constraints on the Data Source and Data Sink.

CA4-111: The transition from Pipelined Mode to Combined Mode **shall** be governed by Figure 267 Data Source Transition from Pipelined to Combined Mode on page 1258 and Figure 268 Data Sink Transition from Pipelined to Combined Mode on page 1259.

CA4-112: The Data Source **shall** complete any RDMA Writes that are In-Process, and issue all RdmaWrCompl messages before sending a ModeChange message.

This ensures that RDMA Writes will not be used in Combined Mode.

CA4-113: If the Data Source has incomplete SinkAvail advertisements, the Data Source **shall** discard them and re-issue all Unprocessed or In-Process SrcAvail messages, if there are any.

Normal Combined Mode rules apply, for example, only one SrcAvail message may be outstanding at any one time and it must contain ULP payload. If the Data Source has no incomplete SinkAvail advertisements, but a SinkAvail advertisement is received before the acknowledgement for the ModeChange message, it shall ignore the SinkAvail message (but process flow control information, etc.) and re-issue all outstanding SrcAvail messages according to Combined Mode rules. In either case, if any more SinkAvail messages arrive after the initial discard of SinkAvail message(s), the Data Source shall ignore these SinkAvail messages (but flow control information, etc., shall be Processed normally).

Normal Data Sink behavior in Pipelined Mode requires it to drop any SrcAvail messages (but process flow control information, etc.) if there is a SinkAvail outstanding.

CA4-114: Thus the only behavior the Data Sink **must** follow when transitioning to Combined Mode after receiving the ModeChange message is to invalidate local state associated with any outstanding SinkAvail messages.

As mentioned previously, the Data Source will ensure that any outstanding SrcAvail messages will be re-issued.

A4.9.5 STATE MODE TRANSITION SUMMARY

Table 359 Data Source Mode Transition Events on page 1260 and Table 360 Data Sink Mode Transition Events on page 1260 summarize the events and consequent actions for the Data Source and Data Sink, respectively. Advisory input to the Mode Master to change modes is not meant to be exhaustive. The Mode Master may change modes at any time, possibly for reasons beyond the scope of this specification. The tables only list events which may cause a mode transitions or may be an event which is handled uniquely in a specific mode.

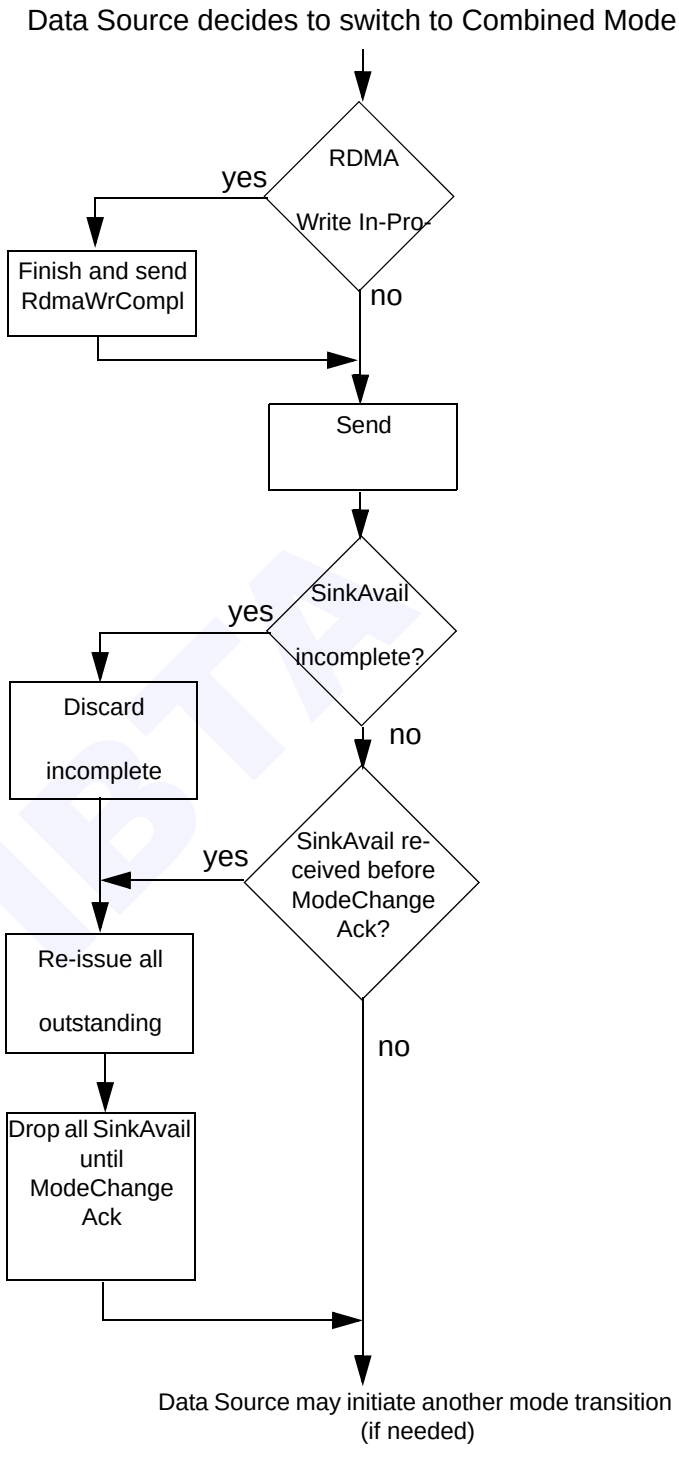


Figure 267 Data Source Transition from Pipelined to Combined Mode

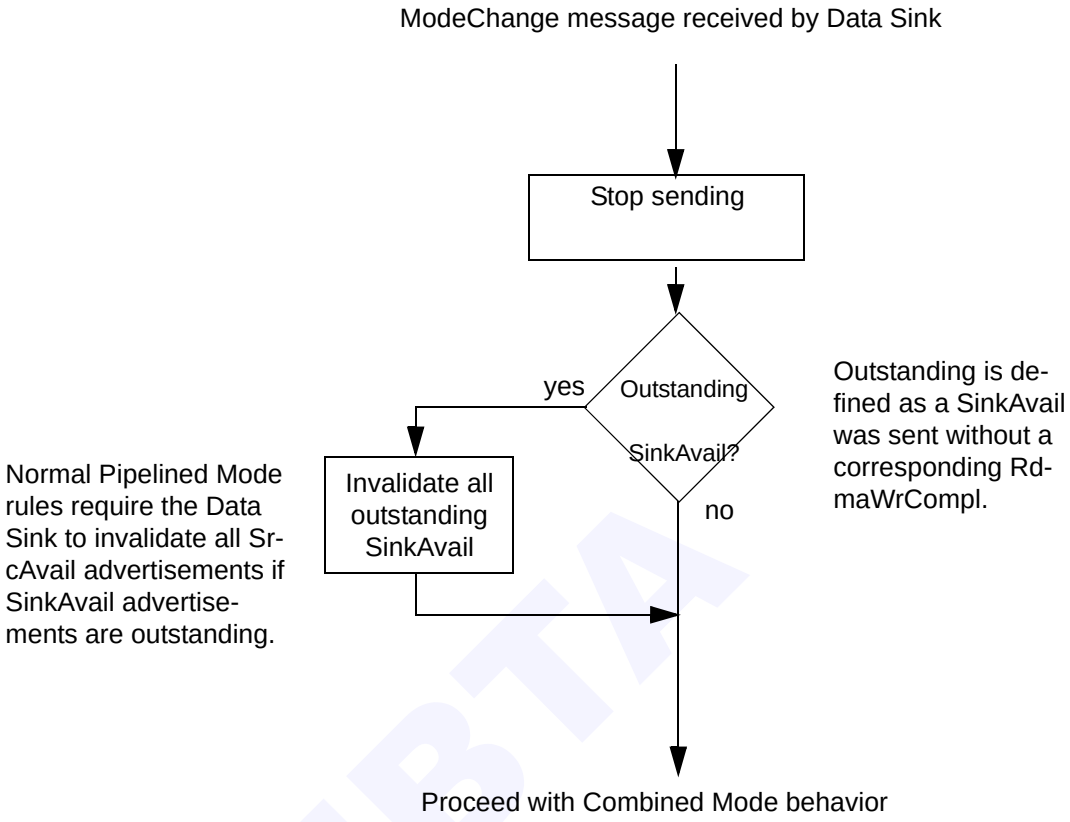


Figure 268 Data Sink Transition from Pipelined to Combined Mode

.

Table 359 Data Source Mode Transition Events

Data Source Mode	Event	Action/Transition
Combined (master)	Receive REQ_PIPE=1 in an RdmaRd-Compl message	Advisory: May decide to transition to Pipelined Mode.
	Data Source decides to change modes to Buffered	If outstanding SrcAvail, wait for RdmaRd-Compl or SendSm from Data Sink, then send ModeChange message
	Data Source decides to change to Pipelined Mode	Change to Pipelined Mode and send a ModeChange message
	Receive SendSm	Advisory: May decide to transition to Buffered Mode.
	Receive SinkAvail	Can happen if just transitioned from Pipelined Mode. See Figure 267 Data Source Transition from Pipelined to Combined Mode on page 1258 for processing details.
Buffered (slave)	Receive ModeChange message with S=1 and Mode=COMB_MODE	Immediately transition to Combined Mode.
Pipelined (master)	Receive SendSm	Advisory: May decide to transition to Combined Mode.
	Data Source decides to change to Combined Mode	Change to Combined Mode and send a ModeChange message. See Figure 267 Data Source Transition from Pipelined to Combined Mode on page 1258 for processing details.

Table 360 Data Sink Mode Transition Events

Data Sink Mode	Event	Action/Transition
Combined (slave)	Receive ModeChange message with S=0 and Mode=BUFF_MODE	Immediately transition to Buffered Mode.
	Receive ModeChange message with S=0 and Mode=PIPE_MODE	Immediately transition to Pipelined Mode.
Buffered (master)	ULP receive buffer is posted	Advisory: If the receive ULP buffer is suitable for RDMA, the Data Sink may choose to transition to Combined Mode.
	Data Sink decides to change to Combined Mode.	Send ModeChange message.
Pipelined (slave)	Receive ModeChange message with S=0 and Mode=COMB_MODE	Immediately transition to Combined Mode. See Figure 268 Data Sink Transition from Pipelined to Combined Mode on page 1259 for processing details.

A4.10 SOCKET DUPLICATION

When a socket exists in one address space and is then accessed in a different address space (on the same peer), the socket needs to be duplicated into the second address space. Note that if two threads are

accessing the socket in the same address space, socket duplication is not required.

Performing socket duplication in user-mode imposes certain restrictions because socket state cannot be shared between the address spaces. In fact, in the context of InfiniBand networks available today, the socket can only exist in one address space at a time (since HCAs are not required to support sharing queue pairs between multiple address spaces).

Because of these restrictions, SDP allows only one address space at a time to execute operations that either transfer data or change state for an underlying shared socket. Address spaces dynamically swap control of the underlying socket, as needed, to execute requested operations. The SDP socket duplication procedure serializes operations that different address spaces request on a shared socket. The procedure waits for all In-Process operations to complete before swapping control of an underlying socket to another address space. Logically, the procedure takes control of the underlying socket away from the controlling address space as soon as a non-controlling address space requests an operation on that socket. After control is taken away, the procedure treats the original controlling address space like a non-controlling address space if the original controlling address space requests operations on that socket. In this way a socket may transition back and forth between controlling address spaces based on ULP behavior.

SDP enables socket duplication by bringing the connection to a consistent state, closing the InfiniBand connection, handing the state to the new controlling address space, and then creating a new reliable connection in the new address space. Note that after the connection is suspended and then restarted on a new InfiniBand connection, the connection by definition does not have any outstanding SinkAvail or SrcAvail advertisements. Any incomplete SinkAvail or SrcAvail advertisements were effectively canceled during the transition to a new connection.

A4.10.1 IMPLEMENTING SOCKET DUPLICATION

CA4-115: An SDP implementation **shall** support responding to a socket duplication request using the procedure defined for the remote peer in section A4.10.1.1.

oA4-2: If SDP socket duplication initiation is supported, an SDP local peer **shall** employ the procedure defined in section A4.10.1.1 when initiating socket duplication.

A4.10.1.1 SOCKET DUPLICATION PROCEDURE

1) The SDP implementation in the non-controlling address space **shall** use the appropriate verbs to accept incoming connection requests at

a Service ID in one of two formats (see the Application Specific Identifiers Annex for additional information):

- 0x0000 0000 0001 XXXX, where 0xXXXX is the TCP destination port number. This may or may not be the original TCP port number specified during the initial connection setup. Note that if this format is used, the connection request should not be completed unless the source IP address, destination IP address, and source TCP port number contained in the Hello message match the expected value. Checking just the Service ID (i.e. destination TCP port number) is not sufficient in some implementations.
- Local OS Administered Service IDs.

The mechanism which enables the Service ID to be conveyed to the controlling address space is outside the scope of this specification.

2) The SDP implementation in the controlling address space **shall** wait for all In-Process data transfer operations to complete, then it **shall** send a SuspComm message to the remote peer to request a suspension of the session. This SDP message also contains the Service ID received from the non-controlling address space. The remote peer **shall** connect to this Service ID when resuming communication (step 4). The local peer **shall** send no more SDP messages, nor perform any RDMA operations from the controlling address space, after sending the SuspComm message.
3) Upon receiving the SuspComm message, the remote peer **shall** wait for all In-Process data transfer operations to complete, then **shall** send a SuspCommAck message indicating that the session is suspended. This peer **shall not** send any more messages or perform any RDMA operations until a new connection is set up (step 8).
4) The remote peer **shall** wait for completion of the send work request for the SuspCommAck message, then close the InfiniBand connection. Before sending the REQ CM MAD, the remote peer **shall** use the address resolution procedure to determine the destination address for the REQ MAD; note that this may be different from the original destination address. It **shall** then initiate the new connection to the Service ID received through the SuspComm message. Posting of receive private buffers and the contents of the HH **shall** follow the same rules as connection setup (see section A4.5.1 Connection Setup on page 1218).
5) Once the SuspCommAck message is received, the controlling address space on the local peer **shall** send a signal to the non-controlling address space through some private means outside the scope of this specification. The controlling address space **shall** also send to the non-controlling address space through some private means outside the scope of this specification:

- any buffered receive ULP data

- the remote peer's TCP port number (to ensure the parameter does not change when the socket is re-connected).
- the size of the local receive private buffers.

6) Non-controlling address space **shall** accept the connection request from the remote peer and initialize its state variables for the new connection. The Hello message in the REQ MAD initializes SDP connection state. The non-controlling address space **shall** also verify that the TCP port number is the same as the TCP port number passed from the controlling address space.

7) When both steps 5 and 6 have completed, the (previously) non-controlling address space:

   a) **Shall** send a CM REP MAD with a HelloAck message to the remote peer (see section A4.5.1 Connection Setup on page 1218). The receive private buffer size parameter in the HelloAck message **shall** be the values received from the controlling address space.

   b) **Shall** make received data from the controlling address space available to the ULP.

8) the remote peer **shall** respond by sending an RTU.

9) When connection setup is complete, the local peer **shall** resume normal data transfer. See section A4.5.1.1 InfiniBand Reliable Connection Setup on page 1218 for a description of connection setup completion.

A4.10.1.2 CONFLICT RESOLUTION

CA4-116: If both peers concurrently send each other SuspComm messages, then the accepting peer **shall** disregard the SuspComm message, while the connecting peer **shall** respond to the SuspComm message. The connecting peer **shall** re-send its own SuspComm message once communication is re-established.

A4.10.2 HCA MANAGED FAILOVER

SDP supports Managed Failover by leveraging the Socket Duplication procedure (see section A4.10 Socket Duplication on page 1260). During socket duplication, a change of address space is occurring. In managed failover, the SDP connection may in fact be reestablished using different paths, ports, HCAs or hosts. The original connection in a managed failover scenario is analogous to the controlling address space in socket duplication. The new failed over connection is analogous to the non-controlling address space. Managed failover changes where one end of the connection is situated. Failing over both ends requires two managed failover operations.

The decision to attempt a managed failover must occur before step 1 of the Socket Duplication procedure. How such a decision is made is dependent on policy and outside the scope of the SDP specification.

oA4-3: If an SDP implementation supports managed failover, it **shall** do so using the socket duplication procedure (see section A4.10.1 Implementing Socket Duplication on page 1261) with the following change to Step 1:

- A new endpoint for the failover connection must be chosen. How the new endpoint is chosen is a matter of policy. However, it must be chosen in such a way that the address (see section A4.4 Address Resolution on page 1218) will resolve to the failover port. This address resolution uses the IP addresses from the original connection's Hello message (see section A4.3.2.1 Hello Message (HH) on page 1204). As before, the Service ID chosen for the SuspComm (see section A4.3.5.1 SuspComm Message on page 1217) message should resolve to the failover endpoint.

A4.11 INFINIBAND TRANSPORT LAYER ISSUES

A4.11.1 INFINIBAND MESSAGE REQUIREMENTS

CA4-117: A conforming implementation of SDP **shall** enable RDMA Reads and RDMA Writes on each RC connection.

CA4-118: A conforming implementation of SDP **shall not** use InfiniBand Immediate Data.

A4.11.2 SOLICITED EVENTS

An SDP implementation occasionally needs to stop processing on a half-connection and wait for one of the following messages to arrive before proceeding further:

1) flow control credit update SDP message - flow control credits for the receive private buffer pool have been exhausted, thus it cannot send data, control, and/or RDMA advertisement to the peer;
2) data or RDMA advertisement SDP message - the ULP has indicated its interest in data via a sockets interface select call or receive;
3) RDMA completion (or cancel) SDP message - before it can de-register a send or receive RDMA buffer it must either complete the RDMA transfer or receive a cancel acknowledgement message.

A typical SDP implementation would request completion queue notification and block the ULP process (or thread) until the appropriate message arrives and the notification is delivered. The goal of using the SE (solicited event) bit is to minimize completion queue notification events and corresponding process (or thread) wake-ups when the arriving message does

not match the class of SDP messages that the implementation requires. For example, if the Data Source is waiting for a RdmaRdCompl message to complete a send ULP buffer and there is no local receive ULP buffer posted or local invocation of a sockets interface select on the opposite half-channel (local Data Sink), it should not receive notifications for the opposite half-connection if the peer sends a Data message or SrcAvail RDMA advertisement.

To accomplish this goal, all SDP messages are subdivided into solicited or unsolicited SDP messages.

Solicited SDP messages are those that most likely require immediate attention regardless of ULP behavior at the remote peer and regardless of whether the remote peer is waiting for other (unsolicited) messages. Solicited SDP messages are defined as:

- AbortConn, SuspComm, SuspCommAck, SendSm, SrcAvailCancel, SinkAvailCancel - because it is essential for the sender of these messages that its peer react to them as soon as possible;
- RdmaWrCompl, RdmaRdCompl, SinkCancelAck - because the peer most likely needs to deregister and release RDMA buffers to the ULP upon reception of these messages;
- Data with OOB_PRES or OOB_PEND bit set - because the peer SDP implementation most likely needs to notify the ULP as soon as possible that this message has been received.

CA4-119: An SDP implementation **shall** request the InfiniBand SE bit in the InfiniBand Base Transport Header be set for solicited SDP messages.

Unsolicited SDP messages are those which may require immediate attention by the peer, but only the peer can decide whether or not a notification is necessary - it depends on the ULP behavior or the implementation of the peer. Unsolicited SDP messages are defined as:

- DisConn, SrcAvail, SinkAvail, Data without OOB_PEND or OOB_PRES bit set - because the peer only needs to immediately process these messages when the ULP has issued a sockets interface select or receive request;
- ModeChange, ChRcvBuf, ChRcvBufAck - because after receiving these messages the peer only needs to take action for new messages it generates itself or messages which follow which are solicited SDP messages (e.g., it will not be blocked specifically waiting for these SDP messages).

CA4-120: An SDP implementation **shall not** request the InfiniBand SE bit in the InfiniBand Base Transport header be set for Unsolicited SDP messages.

The SE bit is never applicable for RDMA Reads, and for RDMA Writes is applicable only for those that contain Immediate Data. Since SDP never generates RDMA Writes with Immediate Data, the SE bit is not applicable for RDMA Read or RDMA Write requests generated by SDP.

A4.11.3 KEEPALIVE MESSAGES

The sockets interface provides the ULP with the capability of periodically transmitting messages to the peer (which require an answer) to determine if the peer is still alive (SO_KEEPALIVE). This is referred to as the keepalive feature, and the associated messages are known as keepalive messages. If an SDP implementation supports the keepalive feature, then it must implement this functionality by using zero-length RDMA Writes (see C9-88: on page 305). A zero-length RDMA does not require a valid memory region at the Data Sink; the R_Key and VA may be set to any value and are not checked at the destination.

Initiation of keepalive messages is determined by SDP message activity as well as the passage of a recurring time period known as the *idle period*. The first idle period begins when the SDP implementation determines that the keepalive feature will be enabled on a particular connection. The duration of the idle period is determined by a value called the *idle interval*; once that amount of time has passed, a new idle period begins.

oA4-4: If the SDP implementation supports the keepalive feature, and keepalive is enabled for a particular connection, then the implementation **shall** obey the following two rules regarding the generation of keepalive messages, and the default value of the idle interval:

1) If no SDP messages are sent or received during an idle period, then at the end of that period the SDP implementation **shall** send a zero-length RDMA Write keepalive message.
2) The default idle interval **shall** be 2 hours.

The value for the default idle interval should be configurable.

If a keepalive message completes in error, the QP will transition to the Error state. This is considered an SDP protocol violation, and the SDP implementation shall perform an abortive close. At that time normal InfiniBand clean up processing must occur to reclaim system resources.

Note that the SDP implementation of keepalive differs somewhat from the traditional TCP implementation in two ways:

1) The successful completion of the keepalive message verifies that the IB Reliable Connection is still operational; it does not assess the health of the remote SDP implementation.

2) SDP does not maintain separate keepalive idle and interval timers for a connection; the IB Reliable Connection retry mechanism serves the function that the keepalive interval timer serves in TCP.

A4.12 SDP COMPLIANCE CATEGORY

In order to claim compliance to the Sockets Direct Protocol Specification to the SDP Compliance Category, a product **shall** meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support.

ANNEX A5: BOOTING ANNEX

A5.1 INTRODUCTION

A5.1.1 PURPOSE

This annex provides relevant information necessary to build IBA solutions which are capable of booting their operating system using IB attached devices. It provides implementers with guidance about how various boot methods can be supported, defines boot resolution methods as policies, and provides a descriptive time-line of the booting process.

Selection of booting methods and boot resolution methods are matters of policy, this annex specifies a Boot Management class and an optional Boot Management Agent (BtA) to manage boot parameters in a booting platform. The Boot Management class provides a mechanism for a system administrator to manage the InfiniBand specific booting policies of platforms that contain a BtA. This is done using Boot Management MADs. Booting policies are installation specific, there are no prescribed defaults for boot methods or boot resolution methods.

This annex addresses the considerations for producing IB specific boot code. These topics include:

- How to identify, select, locate, and use IB I/O Controllers and Console services needed for booting.
- Network Boot Models - IB Network and NIC
- How to manage the IB specific behavior of booting platforms throughout the fabric.
- How to boot using the Boot Information Service
- The dependencies the booting platform has on the IB fabric.
- BtA traps and notices to notify the BootManager of asynchronous events within the booting platform.
- The ability of the booting platform to expand the boot environment by loading and executing additional code from a ROM Repository
- The definition and usage of ROM Repositories when the platform is booting.

A5.1.2 GLOSSARY

The following terms are in addition to the terms in Chapter 2: Glossary on page 69.

BIS — Boot Information Service (see Annex A6: Boot Information Service on page 1403).

BIS Locator Records — Locator Record provided by the BIS.

BIS Resolution Method — The method for locating a ROM Repository, a Console object, or an OS Boot Loader by sending a query to the BIS which returns Locator Records back to the booting platform.

Boot — The bootstrap process used by a platform that terminates when the Operating System is loaded. The process begins with either a power on reset or a node reboot (also called Reboot).

Boot Environment — A software environment that controls the booting platform during the boot process.

Boot Information Service — A management service (specified in the Boot Information Service Annex) that provides boot information to booting platforms upon request.

Boot Management — A class of MADs that allow the BootManager to Set (write) and Get (read) attributes that control the behavior of the booting platform.

Boot Management Agent — An optional General Services Agent (GSA) located behind each port of a booting platform that supports InfiniBand Boot Management.

BootManager — A IBA management entity that issues Boot Management Class MADs to Boot Management Agents to manage the behavior of the platform during the Boot process.

BootMgt — Boot Management

Boot Method — Identifies the style or process (Storage, Network) that a boot environment uses to load the OS.

Boot Platform — An endnode that can Boot using the IB fabric and Locator Records provided by the BootManager or the BIS.

Booting Platform — A Boot Platform that is in the process of booting.

Boot Resolution Method — A way that a Booting Platform selects I/O devices, a Console service, or ROM Repository used during the Boot process. This annex defines two boot resolution methods, which are Local Resolution Method and BIS Resolution Method.

BtA — Boot Management Agent.

Console Locator Record — A record that identifies the Console I/O controller (Console IOC) or Console Service the Booting Platform uses during the Boot process.

Extended Boot Environment — A Boot Environment that has been extended by loading additional code.

Firmware — Firmware is the ROM-based software that controls a computer between the time it is turned on and the time the primary operating system takes control of the machine.

IB Network Boot — A way of booting that reads file(s) from a service using Network boot protocols (e.g. IP) over the IB fabric. See Figure 272 on page 1279.

LAN Network Boot — A way of booting that reads file(s) using a LAN adapter or Network Interface Controller (LAN NIC). See Figure 271 on page 1278.

Local Resolution Method — A method for locating a ROM Repository, a Console object or an OS Boot Loader by using non-volatile storage to persistently save the identities of these entities across power cycles.

Locator Record — Attributes that describe the identity of an OS Boot Loader, Console service, or ROM Repository. In some cases, additional information exists within the locator record that can be used by the protocol to assist in finding a particular OS Boot Loader or Console service when multiple object can be accessed. There are 3 types of locator records - OS Locator Record, ROM Repository Locator Record and Console Locator Record.

NVRAM — Non-volatile Random Access Memory. Refers to any type of non-volatile storage.

OS Boot Loader — The term used to describe code located on an I/O device or boot server that when executed by the Booting Platform will load the target OS.

OS Locator Record — A Locator Record that points to the device that contains an OS Boot Loader used to load the OS into the Booting Platform.

Local-Persistent Locator Records — Boot information that a Boot Platform saves in local, non-volatile memory to use when the platform Boots. A persistent LocatorRecord local to the booting platform is accessible across power-cycles.

Reboot — To restart the boot process. See Boot.

ROM Repository — A non-volatile storage cache on a managed I/O unit that contains images such as device drivers and Extended Boot Environment code.

ROM Repository Locator Record — A Locator Record that points to a ROM Repository containing images such as device drivers and Extended Boot Environment code.

SCSI RDMA — SCSI RDMA Protocol. See SRP.

SRP SCSI RDMA Protocol - An I/O storage protocol defined by ANSI T10 that may be used for the purposes of booting the platform.

Storage Boot A way of booting that reads blocks of information from a storage device such as a hard disk drive.

A5.1.3 OVERVIEW

Booting platform is a conceptual term that describes an entity which starts with an elementary operating environment[34] herein called the boot environment and uses I/O functions to load a more sophisticated operating environment. I/O functions might involve devices and services attached via the IB fabric. A booting platform contains one or more CA's.

This annex describes the booting framework that allows a booting platform to use an IB fabric to boot.

In general, a booting platform needs IB specific information to boot if it is to use the IB fabric in its boot process. That information may be stored locally on the booting platform or provided dynamically by a Boot Information Service (BIS) each time the platform boots. The Boot Information Service Annex specifies BIS operation, which provides the means for the booting platform to query the BIS when it needs boot information.

As shown in Figure 269, this annex defines two management entities; the BtA and the BootManager. These are conceptual entities that send and receive Boot Management Class MADs to affect the booting process (refer to section 13.3 Managers, Agents, and Interfaces on page 713). These MADs are not part of the platform's booting process and are not meant to describe the implementation of the booting platform. The BtA is an optional General Services Agent (GSA) located behind each port of a booting platform that supports InfiniBand Boot Management.

34. A software environment that is not as feature rich as current vintage Operating Systems. For example Firmware (e.g. BIOS) is considered an elementary operating environment.

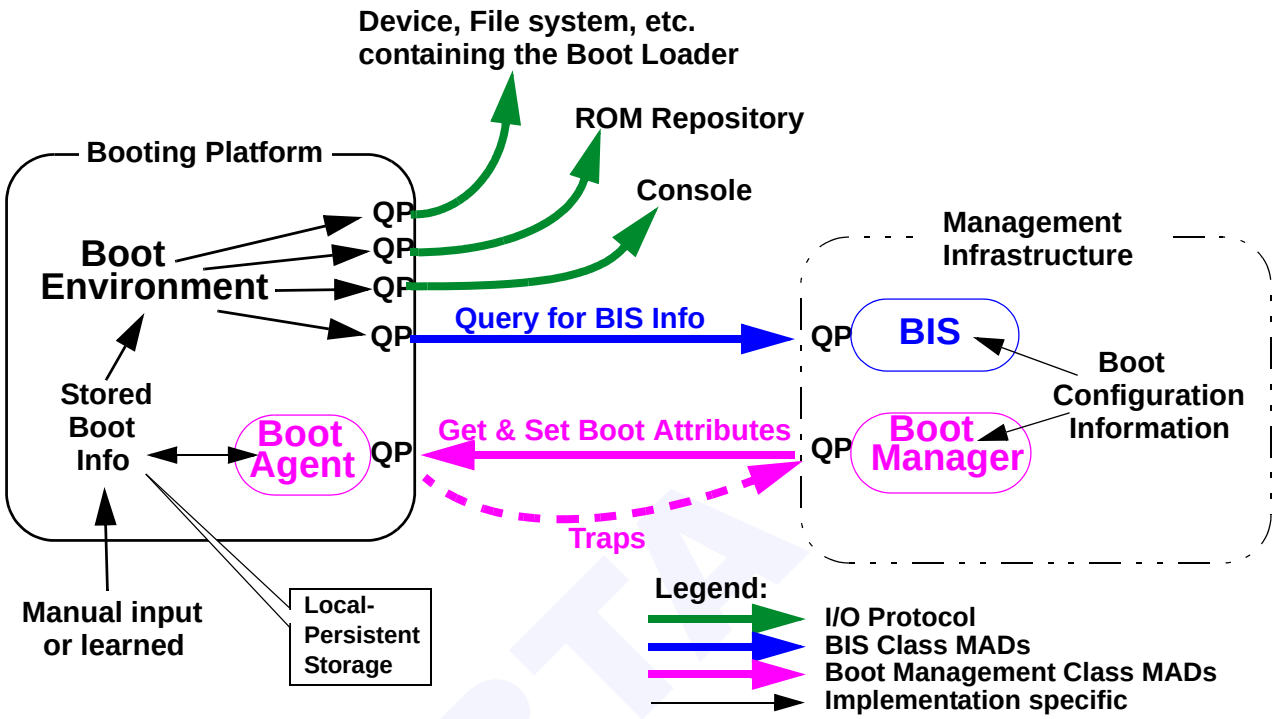


Figure 269 Booting Framework

Figure 269 describes a booting platform containing a Boot Environment, a BtA and persistent storage. A BootManager uses Boot Management class MADs to communicate with the BtA. The Boot Environment through firmware can access the BIS using BIS class MADs. Both Boot Management and BIS class MADs serve the booting platform by providing information used by the platform during the boot process.

The booting framework also supports various phases of booting, providing the booting platform with locator records that identify:

- ROM Repositories that contain additional code to update or expand the boot environment or contain device drivers needed to access the OS Boot Loader.
- A Console Service that the booting platform can use to input information manually and output status.
- The location for the platform's Boot Loader. This includes the source for the installation program, such as when a new platform needs to boot an Install program that installs the OS. A platform boots an Install program just like it boots the OS - by executing what it thinks is a boot loader.

- The destination where the Install Program can place the platform's OS Boot Loader. Thus, if the Install is successful, then the location becomes a source for the platform's OS Boot Loader.

Locator Records (see A5.6.5 "Persistent Locator Records" on page 1327 and A5.6.5.5 "RecordFunction" on page 1331) contain information for a platform to identify a device from which to boot an OS and to locate either a Console device or service. Platforms that do not have Locator Records in non-volatile storage need to query the BIS for Locator Records or wait to be configured with boot information, such as by a BootManager. Proprietary methods to read or write Local-Persistent locator records are out of the scope of IBA. Regardless of how Locator Records are set, they can be read by a BootManager if a BtA is present on the boot platform.

The boot environment (e.g. BIOS) controlling the boot platform after power on reset, reset, or reboot might or might not contain a BtA. After the OS is loaded it too might or might not contain a BtA. The Capability of the BtA in the boot environment may be different than the Capability of the BtA after the OS is running. When the BtA is not running, the BootManager does not have the ability to change local-persistent locator records, timeouts or receive Boot Management Traps/Notices from the booting platform.

A booting platform includes any node booting over IB that contains a CA. This includes compute nodes, I/O nodes, and routers. For example, an I/O unit might import its operating environment from a device or server attached by the IB fabric. A switch or a router may also boot from the fabric and be considered a booting platform. However, booting capability is generally associated with how a compute node loads its operating system (OS).This annex describes the IB booting framework with respect to a booting platform loading its OS.

A booting platform might contain multiple CA's and each CA might contain multiple ports. There are no architectural limitations on which CA or which port can be used for booting. Thus, the selection of the CA and the selection of a port on the CA becomes a decision based on the location of the boot device, port priority, and booting policy. If the boot platform can access a boot device using multiple ports, then the booting platform makes the determination of exactly which port it uses.

Platform booting environments vary based on processor type, platform vendor, etc. Some well-known firmware boot environments include BIOS, IEEE1275, EFI, PA-IODC and there are others. Each of these booting environments provide interfaces for supporting different I/O technologies, such as IBA. Thus, there is some form of boot environment specific code that interfaces the boot environment to the I/O technology. In our case, this is IB specific boot code, as conceptually illustrated in Figure 270.

Figure 271 also illustrates the IB I/O boot model (e.g., I/O controller controlling storage devices) and Figure 272 illustrates the IB Network boot model. Both I/O and services are used in most booting methods. For example, resolution of I/O devices require using certain subnet services such as BIS and make use of storage or network protocols such as SRP.

These considerations vary depending on the Boot Method and the Boot Resolution methods supported. The most pervasive booting methods are storage and network. In addition, most boot environments desire some form of console service and that service can also be provided via IBA.

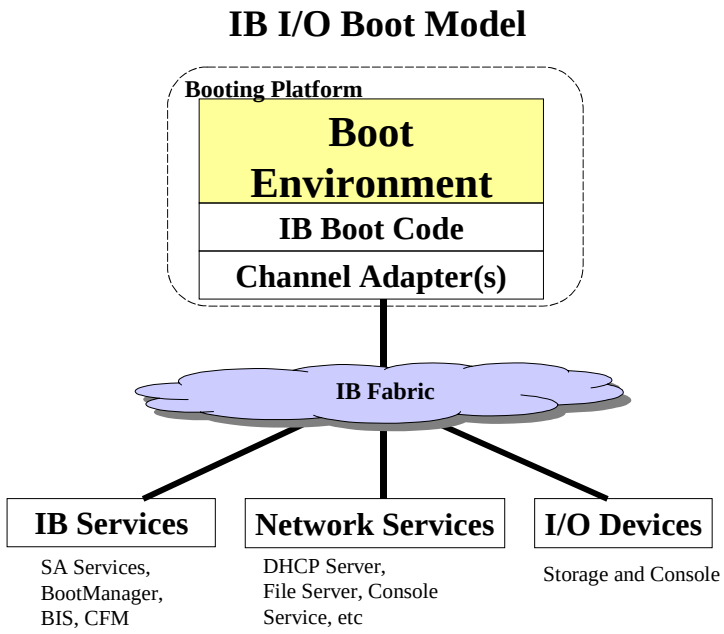


Figure 270 IB I/O Boot Model

A5.1.4 CONSOLE

Typically booting environments use console services to display booting progress, to make booting selections, to input boot parameters, and to troubleshoot or recover from failed boot attempts.

Console service is independent of any boot method. There are two key considerations in supporting console service over IB fabric. The first is the selection of the console service and the second is the console protocol. Naturally, both the booting platform and the console service support the same console protocol. IBA defines a console protocol (see Annex A2: Console Service Protocol on page 1140) and while it is not the only possible console protocol, it is the only one specifically addressed in this

annex. All other console protocols are considered proprietary. The IB booting framework supports proprietary protocols, but does not address issues with implementing them or determining if a booting platform is compatible with a proprietary console protocol.

The IB console service can be provided by a server or by an I/O controller. This annex describes how the IB boot code can locate the server or I/O controller that provides console service.

Normally there is an affinity from a booting platform to a particular console service or device using either local-persistent or BIS Locator Records. If this affinity fails then an exhaustive search for a console might be performed. For a console device the hunt begins when the platform queries all I/O units (IOU) that the booting node has a path to using DevMgtGet(ServiceEntries) and scanning for the well-known service name of Console.IBTA. A console service is found by querying Subnet Administration (SA) with SubnAdm(ServiceRecord) also scanning for the well-known service name of Console.IBTA.

A5.1.5 STORAGE BOOT METHOD

The predominant booting method is booting from a fixed-length random block storage device. Other storage devices such as sequential (tape) storage and variable-length block storage are also viable. The IB booting framework does not differentiate between various storage device types.

Typically the booting environment reads blocks of data from a storage device. For the IB I/O model, that storage device is accessed by an I/O controller (IOC) that is part of an I/O unit (IOU).

There are two key considerations in supporting a storage method over the IB fabric. The first is the boot resolution method which selects the IOC. The second is the boot method (i.e. the storage protocol) used after selecting to the IOC. Naturally, both the booting platform and the IOC support the same I/O protocol. ANSI T10 defines a SCSI protocol specifically suited for IBA (see Annex A1: I/O Infrastructure on page 1121). It is called SCSI RDMA protocol or SRP. While SRP is not the only possible storage protocol, it is the only one specifically addressed in this annex. All other storage protocols are considered proprietary. The IB booting framework supports proprietary protocols, but does not address issues with implementing them or determining if a booting platform is compatible with a proprietary protocol.

This annex describes how IB boot code can locate the I/O controller for a storage device and determine the I/O protocol using Locator Records. Through the use of multiple Locator Records, the IB booting framework also addresses ways to establish an affinity with a storage controller as well as the ability to support redundant storage controllers.

A5.1.6 NETWORK BOOT METHOD

The Network boot method is considered more complex than storage because it involves multiple services and multiple protocols. There are a number of network booting methods that range from simply identifying the boot server and then extracting the boot file to a complex, service rich network environment.

There are multiple ways for the booting platform to access network based services. Two of them are as follows.

- One way is through a LAN Network Interface Controller (LAN NIC) as illustrated in Figure 271. In this case the boot environment (BIOS, etc.) needs a LAN driver that controls the LAN NIC so it can send and receive network packets through the NIC. The I/O protocol between the LAN driver and the LAN NIC is not specified by IBA. Boot resolution is very similar to that for a Storage Boot method because the booting platform only needs to locate the I/O controller (LAN NIC), make a connection to the I/O controller, and then use standard IP protocols over an IB connection.
-

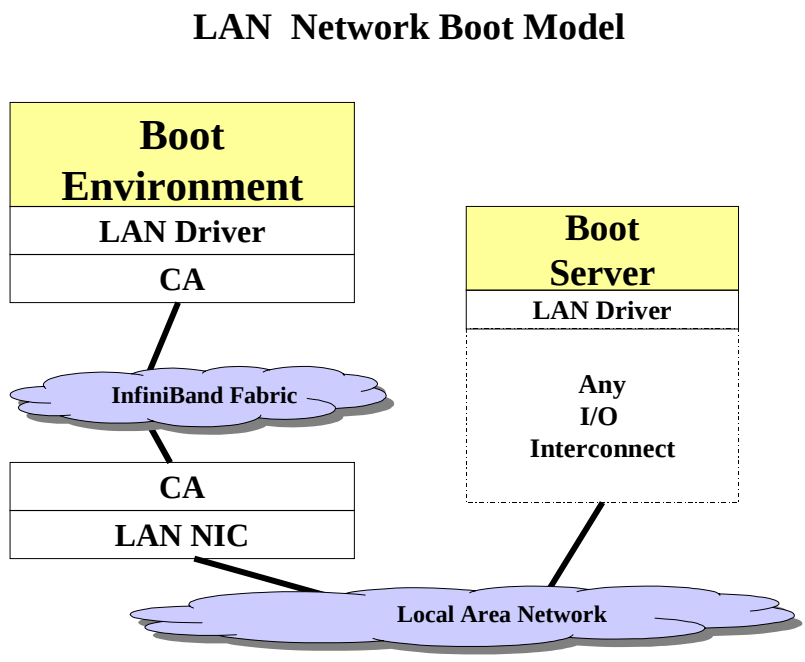


Figure 271 LAN Network Boot Model

- The second case is where the Boot Server is attached to the IB fabric and the IB fabric is the network as illustrated in Figure 272.

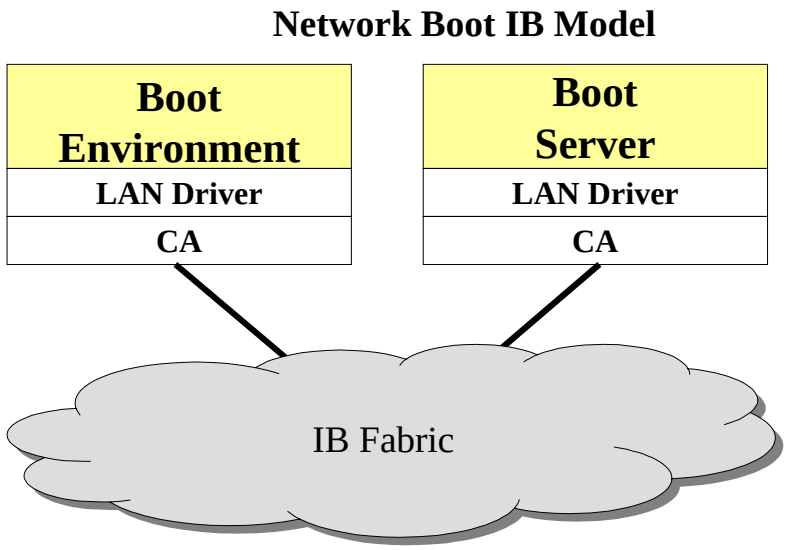


Figure 272 IB Network Boot Model

The IB network boot model does not require a LAN NIC. In this case, the boot environment (BIOS, etc.) needs a LAN driver that can send and receive IP packets over the IB transport.

A5.1.7 BOOT ENVIRONMENT

The boot environment is defined as the software environment (including the BtA) running before the booting platform's OS is operational. The boot environment makes use of CA's on the IB subnet that provide I/O and subnet services needed to boot.

Console Service - Boot firmware, OS loaders and low-level applications (diagnostics, recovery) often assume the presence of a console, which is an abstraction for a pair of input and output devices used for interaction with an administrator.

This Annex describes how consoles are discovered. The Console Annex defines a wire protocol for consoles, which is suitable for use with console devices as well as console services.

Power Management - The management infrastructure provides for power management including the ability to power down and wake a node. Booting principles assume that I/O units are alive and active. The booting host should take into consideration that there may be a delay from the time that it starts to send MADs to an I/O unit, until the I/O unit is capable of responding. This delay could include the time for the power manager to

respond to a trap and the time for the I/O unit to initialize. Currently this time is not bounded, but is expected to be in the order of seconds.

It is recommended that I/O units that take significant time to initialize, especially those supporting power management and wake-on-fabric (see Volume 2, 12.7 “Power Management Wake-up”), be able to respond with DevMgtGetResp(PortClassInfo) as soon as possible.

A5.1.8 MANAGING THE BEHAVIOR OF A BOOTING PLATFORM

Subnets range from very simple to very complex and the selection of the appropriate Boot Resolution Methods is dependant on subnet complexity. Also, booting platforms with multiple ports can belong to multiple subnets with multiple BootManagers and more than one BIS.

A5.1.8.1 BOOT RESOLUTION METHODS

The IB booting framework defines two Boot Resolution Methods and provides two ways to manage them and allows for IB Network Boot.

One Boot Resolution Method, called the Local Resolution Method, simply remembers (through non-volatile storage located on the booting platform) information (i.e Locator Records) required to locate devices or services for the purpose of booting a platform over the IB fabric. Using local-persistent Locator Records provides the boot platform a relatively static way to identify boot devices or services without depending on additional fabric services such as BIS.

Another Boot Resolution Method, called Boot Information Service (see Annex A6: Boot Information Service on page 1403) also resolves devices and services used for booting. It provides a booting platform with the identity of devices and services it needs to boot over the IB fabric. Using the BIS in this manner provides both an adaptive and redundant boot capability that scales with subnet size. A booting platform can query the BIS for any particular attribute and the BIS returns the requested attributes. The attributes returned by the BIS to the booting platform are identical to the R/W components of the equivalent Boot Management attributes (PlatformBootInfo, PortBootInfo, RomRepositoryLocatorRecord,ConsoleLocatorRecord and OsLocatorRecord) written by the BootManager to the BtA.

Boot resolution methods vary from site to site and between platform vendors. Subnet management providers are likely to support both the Local and BIS Resolution methods. Therefore, the Boot Management class provides the means to specify which boot resolution method to use and in what order to use them when both are selected. Local resolution allows the booting platform to maintain a list of Locator Records for one or more devices. The BIS provides a more dynamic and centralized mechanism to manage Locator Records.

Using the locator records, the boot environment can locate a specific OS boot loader, console objects, and ROM Repositories within the IB fabric.

A5.1.8.2 ROM REPOSITORY

To accommodate extending the boot environment and 3rd party drivers, the IBA booting framework provides for ROM Repositories. See A5.12 “ROM Repository” on page 1365. The booting platform can discover if there are Option ROM images designed for that particular boot environment. It also provides the means (i.e. protocol) to download the desired ROM image to the booting platform. Of course what happens to the image (i.e., how the code is linked to the boot environment and how it gets executed) is outside the scope of this document.

A5.1.8.3 BOOT ENVIRONMENT EXTENSION

A platform's boot environment code might provide full functionality or might contain a minimum amount of code that has the capability of being extended. When shipped with code that can be extended, the booting platform needs the ability to access additional code located in another node’s ROM Repository. This extended code together with the co-resident code of the booting platform provides an extended boot environment. In general, the booting platform powers on, locates a ROM repository using either Local or BIS supplied ROM Repository Locator Records, and then downloads additional boot environment code, which provides a richer set of services or functions, including enhancing the BtA itself.

When a power on reset or reboot takes place, the booting platform uses saved platform boot information to determine the method (BIS or Local-Persistent) that it uses to locate and connect to a ROM Repository.

A5.1.8.4 PROPRIETARY DRIVER LOAD

Many booting environments support loading additional code such as 3rd party drivers. This allows the boot environment to import additional code such as proprietary I/O protocols for existing boot methods. This includes extending the boot environment on the booting platform to cover new boot methods.

A5.1.9 SUBNET INITIALIZATION

The Master SM has the task of stabilizing the fabric (see 14.1 Subnet Management Model on page 794”). The Master SM initializes the fabric (node discovery/configuration, topology discovery, and routing table configuration). A booting node might also be the Master SM and thus performs these functions before it can boot. A booting platform that does not contain SM functionality waits until the fabric is stable (PortState = Armed or Active) before the port is allowed onto the IB subnet. A booting platform that does contain SM functionality waits until the subnet is initialized, optionally loads code images from a ROM Repository, and then finds its OS boot loader on the fabric.

Power is often applied to many platforms at the same time, requiring special consideration. A number of factors (CPU speed, memory testing, other diagnostics etc.) allow nodes in the fabric to become SM's at different points of time with a default or persistent recollection of SM Priority. The Master SM (MSM) will switch from node to node until the highest priority SM is on the fabric. From a booting platform's perspective it means that IB paths may be reassigned at any time after the first MSM sets the routing tables/LIDs until the MSM with the highest priority wins the SM discovery process. Subnet management is a dynamic process that responds to topology changes. SM election is completely asynchronous to the boot process, thus the boot environment should tolerate MSM changes.

A5.1.10 BOOT/REBOOT

Booting takes place when the platform is powered on and the firmware running on the platform loads the operating system. Generally, rebooting follows this same process except that power is not cycled.

A5.2 BOOTMANAGER

The Boot Management class provides mechanisms (methods and attributes) to enable a BootManager to set and retrieve boot information from platforms supporting Boot Management.

A5.2.1 GENERAL OPERATION

The BootManager configures a boot platform for the subnet's booting policy using attributes listed Table 366 Boot Management Attribute Summary on page 1293. The booting platform uses these attributes to guide the booting process.

A BootManager communicates with the BtA as shown in Figure 273. The BootManager sends a message (BootMgtSet() or BootMgtGet()) to the BtA and the BtA responds with a BootMgtGetResp(). The BtA can also send Traps to the BootManager and the BootManager can retrieve Notices (not illustrated) from the BtA using BootMgtGet(Notice) and BootMgtSet(Notice).

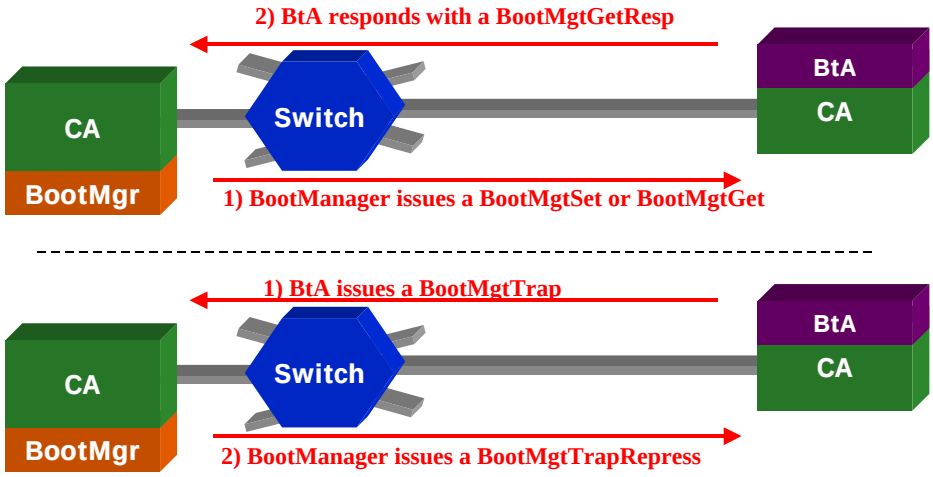


Figure 273 BootManager/BtA Relationship

Boot methods and resolution policies are dictated by the user or computing facility policies. There is no single boot method nor boot resolution policy that is valid for all. In fact, a boot method and resolution strategy that might be ideal in one environment (such as a small single host static topology) is undesirable and sometimes prohibited for another environment (such as a large dynamic facility, clusters, etc.). Additionally, as a site continues to evolve, it may become necessary to vary those strategies over time. A BootManager is also useful to update a platform's boot attributes when subnet configuration changes might otherwise leave the booting platform unable to boot.

The system administrator uses the BootManager as a mechanism to manage the booting policy of each boot platform in order to:

- Assure that when a booting platform is installed, it can be configured to follow the booting policies of the facility. For example, setting Platform Boot Information, Port Boot Information, and Locator Records convey that information to the booting platform via BootMgt MADs.
- Reconfigure each booting platform's boot policy when the booting policy changes.
- Intervene when configuration changes or failures prevent a booting platform from booting.
- Select the BIS port priority for when a booting platform attaches to multiple subnets (see Table 372 PortBootInfo Attribute on page 1321).

- Set the priority of the local-persistent Locator Records. This allows the BootManager to pass the booting platform multiple Locator Records and indicate the order by which each should be tried.

One way for the BootManager to determine if a BtA resides on a booting platform is to query the SA using SubnAdminGet(PortInfoRecord) for any port on the platform. Using CapabilityMask:IsBootManagementAgentSupported from the PortInfoRecord query returned by the SA, the BootManager determines if the BtA exists. If so, the BootManager can determine if the BtA supports either Traps or Notices by testing BootMgtGetResp(ClassPortInfo:CapabilityMask(0)) = 1b and BootMgtGetResp(ClassPortInfo:CapabilityMask(1)) = 1b respectively. Traps and Notices from the booting platform allows the BtA to notify the BootManager of problems that prevent the platform from booting and alerting the system administrator if necessary.

Third parties can register to receive Boot Management Traps and Notices initiated from a port that contains a BtA by issuing a BootMgtSet(InformInfo) to the BootManager. When a trap event occurs on the booting platform, the platform issues a trap to the BootManager. The BootManager forwards the Trap to the third party through the BootMgtReport(Notice) method and attribute. For more information on Boot Management Traps and Notices see A5.6.7 “Traps and Notice Queues” on page 1336]

A5.2.2 DETECTING NEW BOOT PLATFORMS

The BootManager needs to be aware of booting platforms entering and exiting the subnet. The BootManager can subscribe with the SA for in service and out of service Traps (Trap number 64/65). Using traps 64/65 the BootManager is kept informed of the nodes that are online and reachable by the BootManager.

CA5-1: The BootManager shall check for new BtAs by subscribing to SA for trap 64.

In addition to being aware that new booting platforms have entered the subnet the BootManager should be aware that the capability of a booting platform are not static and may change and the BootManager may periodically query each booting platform by querying PlatformBootInfo:Capability. The booting platform’s Capability may change whenever a new BtA is loaded for any reason including expansion of the boot environment and the OS loading a new BtA.

A5.2.3 MULTIPLE BOOT MANAGERS

When multiple BootManagers access the same platform, then access to a booting platform should be serialized by the BootManagers to prevent race conditions that result when multiple BootManagers are reading and writing attributes on a platform at the same time.

A5.2.4 PROTECTING THE BTM_KEY

When BtM_KeyInfo:BtM_KeyProtectBits are non-zero, the BootManager needs to protect against unauthorized access by sending each BtA a BootMgt MAD with a valid MADHeader:BtM_Key before BtA's BtM_Key lease period expires.

CA5-2: When the BootManager sets a non-zero BtM_Key and a non-zero BtM_ProtectionBits in a booting platform, the BootManager shall reset the lease period counter by sending a BootMgt MAD with MADHeader:BtM_Key equal to BtM_KeyInfo:BtM_Key at an interval less than the lease period.

A5.2.5 EVENT REPORTING

The BootManager needs to collect Notices from booting platforms that support Notices.

oA5-1: If the BootManager supports event subscriptions, then the BootManager shall periodically query the BtA notice queue and shall generate a BootMgtReport() for each notice.

A5.2.6 SA ADVERTISEMENT

The BootManager advertises it's location using the ServiceRecord specified in 15.2.5.14 ServiceRecord on page 895. Nodes that choose to subscribe to Traps and Notices can find the BootManager by querying the SA.

CA5-3: The BootManager shall register with the SA via the SubnAdmSet(ServiceRecord) using the <ServiceName>="BootManager.IBTA". If the BootManager is configured for multiple partitions, it shall register once for each partition.

The values for the other ServiceRecord parameters are implementation dependent.

If for any reason a BootManager cannot advertise it's location with the SA using the ServiceRecord then 3rd party nodes will not be able to subscribe to Traps and Notices from booting platforms.

CA5-4: If the BootManager cannot register with the SA then it shall not send Boot Management class MADs to a BtA.

The BootManager is responsible for periodically renewing its lease with the SA to prevent the SA from dropping its ServiceRecord from the SA database (see 15.2.5.14 ServiceRecord on page 895 and 15.2.5.14.3 ServiceLease on page 898).

CA5-5: The BootManager shall renew its registration lease by reregistering with the SA before its service lease expires.

A5.3 BOOTAGENT

A BootAgent (BtA) provides the means for a BootManager to remotely manage the platform's boot attributes. At a minimum, it provides a way for the BootManager to read the platform's booting attributes and optionally provides the means for setting them.

The BtA behind each port allows the BootManager to:

1) Get() (read) the booting platform's capabilities.
2) Set() locator components (see A5.6.3.4 Boot Record Locator Sources on page 1315) to control resolution methods and the specific order that the platform uses to locate a ROM Repository, the Console, and the OS.
3) Set() time-out components to set the minimum amount of time a booting platform should wait to connect to devices or services required to boot.
4) Set() the local-persistent Locator Records the platform will use to boot and the order by which they will be used when booting
5) Be notified (via Traps or Notices) of events that occur within the booting platform that involve the boot process

Boot Management class attributes with a scope of platform can be read and written by a BootManager through any port with identical results. Boot Management class attributes with a scope of port can be read and written by a BootManager on a particular port. Table 363 specifies the scope of each attribute.

CA5-6: The BtA shall return the same component values on all ports of the booting platform for attributes that have a platform scope (as per Table 363).

A5.4 MAD FORMAT

CA5-7: The datagrams in the Boot Management class shall conform to the MAD format and use as specified in Vol1, 13.4 Management Datagrams and further customized in Figure 274 "Boot Management MAD" on page 1286, and Table 361, "Boot Management MAD Components," on page 1287

.

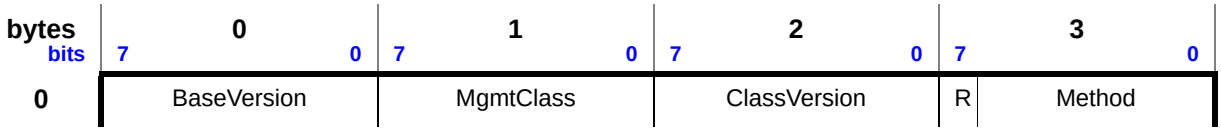


Figure 274 Boot Management MAD

bytes	0	1	2	3
4	Status		Reserved	
8	TransactionID			
12				
16	AttributeID		Reserved	
20	AttributeModifier			
24	BtM_Key			
28				
32	Reserved			
...				
60				
64	BootMgtData			
...				
252				

Figure 274 Boot Management MAD (Continued)

Table 361 Boot Management MAD Components

Component Name	Offset (bits)	Length	Description
BaseVersion	0	1 byte	as per 13.4 Management Datagrams on page 717
MgmtClass	8	1 byte	Management class value = 0x11 indicates Boot Management
ClassVersion	16	1 byte	Version field. Value is set to 1.
R	24	1 bit	as per 13.4 Management Datagrams on page 717
Method	25	7 bits	Boot Management Method, as defined in Table 362 on page 1289
Status	32	2 bytes	The Status field (0:7) is defined in Volume 1, 13.4.7 “Status Field”. Status bits 8-15 are class specific and reserved for future definition. See A5.4.1 “Boot Management MAD Status” on page 1288
Reserved	48	2 bytes	Reserved
TransactionID	64	8 bytes	as per 13.4 Management Datagrams on page 717
AttributeID	128	2 bytes	as per 13.4 Management Datagrams on page 717 and further defined in Table 366, “Boot Management Attribute Summary,” on page 1293.
Reserved	144	2 bytes	Reserved

Table 361 Boot Management MAD Components (Continued)

Component Name	Offset (bits)	Length	Description
AttributeModifier	160	4 bytes	See Table 366 Boot Management Attribute Summary on page 1293
BtM_Key	192	8 bytes	A 64-bit (8-byte) key used to validate Boot Management operations. See A5.6.2 BtM_KeyInfo on page 1295
Reserved	156	32 bytes	Reserved
BootMgtData	188	192 bytes	Attribute data

A5.4.1 BOOT MANAGEMENT MAD STATUS

13.4.7 Status Field on page 731 defines the use of the MAD Status component.

Busy

The BootManager may attempt to alter attributes such as PlatformBootInfo, PortBootInfo, and Locator Records at anytime. If the BtA relies on stable boot parameters during the boot process, then the BtA can reject the Set() command by setting bit 0 of the Status component. For example, while booting, some implementations may expect registers to be stable for the duration of the boot. If the implementation dictates this behavior then the BtA may return the MAD with a Status(0)=1b.

Invalid Field Encode

A Set() or Get() is rejected (i.e. MAD:Status(2:4)=0x3) by the BtA when the attribute, a component of an attribute, or an attribute/component combination cannot be accepted by the BtA. One example of the usage of this status is when the BtA receives a BootMgtSet(RomRepositoryLocatorRecord) and the BtA does not have the capability of setting this attribute.

When a BtA receives a Set() the Read Only (RO) components of the attribute are ignored.

A BootMgtSet() informs the recipient to set values maintained by the recipient according to the values contained in the attribute conveyed in MADHeader:BootMgtData. The components of the attribute shall be set equal to the equivalent values maintained by the recipient after the set has been performed except when the Status returned is non-zero.

By convention, reserved fields must be filled with 0 by the initiator and ignored by the receiver (see 13.4.1 Conventions on page 717).

CA5-8: A BtA or a BootManager that does not support the class version in a request message shall respond by returning the message and setting the R bit, the MAD:Status(2:4) to 0x1 and the ClassVersion set to 1.

CA5-9: A BtA or a BootManager that does not support the method in a request message shall respond by returning the message and setting the R bit and set MAD:Status (2:4) to 0x2.

CA5-10: A BtA or a BootManager that does not support the method/attribute combination in a request message shall respond by returning the message and setting the R bit set MAD:Status (2:4) to 0x3 in the response.

CA5-11: A BtA or a BootManager that does not support a particular value in a Set request message shall respond by returning the message and setting the R bit and set MAD:Status (2:4) to 0x7 in the response.

CA5-12: A BootManager shall ignore the contents of the BootMgtData component if the BtA returns a non-zero status in a BootMgtGetResp().

CA5-13: A BtA shall ignore RO components when processing a BootMgtSet().

Policy Reject

The Boot Manager may reject a BootMgtSet(InformInfo) using MAD:Status(11:8)=0x3 to indicate that the boot manager's policy does not permit subscription to that trap.

CA5-13.2.1: If the Boot Manager rejects a BootMgtSet(InformInfo) because it does not permit subscription to that trap, it shall reject the request using MAD:Status(11:8)=0x3.

A5.4.2 MAD BTM_KEY

The BtM_Key in the MAD:Header is used to validate the source of the BootMgt MADs. See section A5.6.2.1 "BtM_Key General Use" on page 1296 for additional details.

A5.5 BOOT MANAGEMENT METHODS AND ATTRIBUTES

The Boot Management class methods listed in Table 362 are a subset of the common methods described in section 13.4.5 Management Class Methods on page 721.

Table 362 Boot Management Methods

Method	Value (8b=R,Method)	Description
BootMgtGet()	0x01	Request a get (read) of a Boot Management attribute.
BootMgtSet()	0x02	Request a set (write) of a Boot Management attribute

Table 362 Boot Management Methods (Continued)

Method	Value (8b=R,Method)	Description
BootMgtGetResp()	0x81	Response to a BootMgtGet() or BootMgtSet().
BootMgtTrap	0x05	A Boot Management datagram providing the BootManager with asynchronous event notification. Traps are described in 13.4.9. Trap support indicated by BootMgtGetResp(ClassPortInfo:CapabilityMask) bits(0:1) described in 13.4.8.1.
BootMgtTrapRepress()	0x07	A Boot Management datagram notifying the BtA to stop reporting a particular instance of a Trap matching the TransactionID and the Notice attribute.
BootMgtReport()	0x06	BootManager Only, not BtA -Sent from a BootManager to forward Traps/Notices to subscribers.
BootMgtReportResp()	0x86	BootManager Only, not BtA - Response to a BootMgtReport, sent by subscriber to the BootManager.

Table 363 shows the complete list of BootMgt class method and attribute combinations and indicates which are mandatory or optional. Section A5.6 Boot Management Attribute Definitions on page 1293 defines Boot Management attributes.

Attribute Scope identifies the attribute as affecting booting parameters of the port or the entire platform.

Table 363 BtA Method/Attribute Combinations

Attribute Name	Attribute Scope	Methods				
		BootMgt Get	BootMgt GetResp	BootMgt Set	BootMgt Trap	BootMgt Trap Repress
ClassPortInfo	Port	Mandatory			n/a	
BtM_KeyInfo	Platform	Mandatory			n/a	
PlatformBootInfo	Platform	Mandatory		Conditional	n/a	
PortBootInfo	Port	Mandatory		Conditional	n/a	
RomRepositoryLocator-Record	Platform	Conditional		Conditional	n/a	
ConsoleLocatorRecord	Platform	Conditional		Conditional	n/a	

Table 363 BtA Method/Attribute Combinations (Continued)

Attribute Name	Attribute Scope	Methods				
		BootMgt Get	BootMgt GetResp	BootMgt Set	BootMgt Trap	BootMgt Trap Repress
OsLocatorRecord	Platform	Conditional		Conditional	n/a	
NodeReboot	Platform	Conditional			n/a	
Notice	Platform	Conditional			Conditional	

CA5-14: A BtA shall implement the methods and attribute combinations listed as Mandatory in Table 363 on page 1290. and conditionally listed in Table 364 on page 1291.

Table 364 describes the Boot Management Methods and Attributes that

Table 364 BtA Capability Requirements

Method(Attribute)	Condition/Requirement
BootMgtSet(PlatformBootInfo)	If Capability(10),Capability(11) or Capability(12)=1b, then a BtA shall process BootMgtSet(PlatformBootInfo) and BootMgtGet(PlatformBootInfo) and return BootMgtGetResp(PlatformBootInfo).
BootMgtSet(PortBootInfo)	If Capability(8),Capability(10),Capability(11) or Capability(12)=1b, then a BtA shall process BootMgtSet(PlatformBootInfo) and BootMgtGet(Platform-BootInfo) and return BootMgtGetResp(PlatformBootInfo)
BootMgtGet(RomRepositoryLocator-Record), BootMgtGetResp(RomRepositoryLocator-Record)	If Capability(0) or Capability(1) = 1b then a BtA shall process BootMgt-Get(RomRepositoryLocatorRecord) and return BootMgtGetResp((RomRe-positoryLocatorRecord)
BootMgtSet(RomRepositoryLocator-Record)	If Capability(10) = 1b then a BtA shall process BootMgtSet(RomRepository-LocatorRecord) and return BootMgtGetResp((RomRepositoryLocator-Record)
BootMgtGet(ConsoleLocatorRecord), BootMgtGetResp(ConsoleLocatorRecord)	If Capability(2) or Capability(3) = 1b then a BtA shall process BootMgt-Get(ConsoleLocatorRecord) and BootMgtGetResp(ConsoleLocatorRecord)
BootMgtSet(ConsoleLocatorRecord)	If Capability(11) = 1b then a BtA shall process BootMgtSet(ConsoleLocator-Record) and return BootMgtGetResp((ConsoleLocatorRecord)
BootMgtGet(OsLocatorRecord), BootMgtGetResp(OsLocatorRecord)	If Capability(4),Capability(5),Capability(6) or Capability(7) = 1b then a BtA platform shall process BootMgtGet(OsLocatorRecord) and BootMgtGet-Resp(OsLocatorRecord)
BootMgtSet(OsLocatorRecord)	If Capability(12) = 1b then a a BtA shall process BootMgtSet(OsLocator-Record) and return BootMgtGetResp((OsLocatorRecord)

Table 364 BtA Capability Requirements (Continued)

BootMgtSet(NodeReboot)	If Capability(14) or Capability(15)= 1b then a BtA shall process BootMgtSet(NodeReboot), BootMgtGet(NodeReboot) and BootMgtGetResp(NodeReboot) and return BootMgtGetResp(NodeReboot)
BootMgtSet(Notice), BootMgtTrapRepress(Notice)	If ClassPortInfo:CapabilityMask(0) = 1b then a BtA shall process BootMgtTrap(Notice) and BootMgtTrapRepress(Notice).
BootMgtSet(Notice) and repress	If ClassPortInfo:CapabilityMask(1) = 1b then the BtA shall process BootMgtSet(Notice) and BootMgtGet(Notice).

are conditionally supported based on the values found in PlatformBootInfo:Capability and ClassPortInfo:CapabilityMask

CA5-15: A BtA shall implement the methods and attribute combinations listed as Conditional in Table 363 on page 1290 when the conditions listed in Table 364 on page 1291 are true.

Some vendors may implement proprietary means to create or alter R/W components and locator records. This may be done through various means including a console session and is outside the scope of this Annex.

oA5-2: If the Booting Platform supports Local RomRepositoryLocatorRecords then these records shall be readable via a BootMgtGet(RomRepositoryLocatorRecord) regardless of the method used to set RomRepositoryLocatorRecords.

oA5-3: If the Booting Platform supports Local ConsoleLocatorRecords then these records shall be readable via a BootMgtGet(ConsoleLocatorRecord) regardless of the method used to set ConsoleLocatorRecords.

oA5-4: If the Booting Platform supports Local OsLocatorRecords then these records shall be readable via a BootMgtGet(OsLocatorRecord) regardless of the method used to set OsLocatorRecords.

oA5-5: If the BtA supports BootMgtGet(RomRepositoryLocatorRecord) then it shall return the record indicated by the AttributeModifier as specified in A5.6.5 “Persistent Locator Records” on page 1327.

oA5-6: If the BtA supports BootMgtGet(ConsoleLocatorRecord) then it shall return the record indicated by the AttributeModifier as specified in A5.6.5 “Persistent Locator Records” on page 1327.

oA5-7: If the BtA supports BootMgtGet(OsLocatorRecord) then it shall return the record indicated by the AttributeModifier as specified in A5.6.5 “Persistent Locator Records” on page 1327.

Table 365 describes additional methods and attribute combinations that are supported by the BootManager allowing 3rd parties to subscribe to traps and notices.

Table 365 BootManager Method/Attribute Combinations

Attribute Name	Methods		
	BootMgt Get	BootMgt Set	BootMgt Report
ClassPortInfo	x	x	
Notice	x	x	x
InformInfo		x	

A5.6 BOOT MANAGEMENT ATTRIBUTE DEFINITIONS

This section specifies the format of the attributes used for managing booting platforms over the IB fabric.

.

Table 366 Boot Management Attribute Summary

Attribute Name	Attribute ID	Attribute Modifier	Description
ClassPortInfo	0x0001	0x00000000	as per Chapter 13: Management Model on page 709
BtM_KeyInfo	0x0010	0x00000000	BtM_Key information for the platform used to check received Boot Management class MADs.
PlatformBootInfo	0x0020	0x00000000	The PlatformBootInfo attribute allows the BootManager to Get() the booting platforms capabilities and Set() components to control certain boot behavior.
PortBootInfo	0x0021	0x00000000	The PortBootInfo attribute allows the BtA to Get() and Set() the booting platforms BIS priority and time-out values on a port by port basis.
RomRepositoryLocatorRecord	0x0030	0x00000000 - 0x000000FF	A Locator Record that identifies a ROM Repository used to expand the boot environment, BtA, and/or contains proprietary device drivers. The AttributeModifier determines the priority of the record.
ConsoleLocatorRecord	0x0031	0x00000000 - 0x000000FF	A Locator Record that identifies the console to be used by the booting platform. The AttributeModifier determines the priority of the record.

Table 366 Boot Management Attribute Summary (Continued)

Attribute Name	Attribute ID	Attribute Modifier	Description
OSLocatorRecord	0x0032	0x00000000 - 0x000000FF	A Locator Record that identifies the device that the booting platform will use to bootstrap the OS. The AttributeModifier determines the priority of the record.
NodeReboot	0x0040	0x00000000	Allows the BootManager to reboot a platform.
Notice	0x0002	0x00000000	Attribute supporting Traps and Notice Queues
InformInfo	0x0003	0x00000000	Attribute to subscribe to Boot Management Traps. InformInfo is not used by the BtA.
Reserved	Other values reserved	0x00000000 - 0xFFFFFFFF	Reserved

CA5-16: The BtA shall return MAD:Status(2:4) = 0x3 when an invalid AttributeID or AttributeModifier is specified as per Table 366.

A5.6.1 CLASSPORTINFO

Table 367 Boot Management ClassPortInfo:CapabilityMask

Component	Access	Offset (bits)	Length (bits)	Description
ClassVersion	RO	0	8	Current supported BootMgt class version = 0x01
CapabilityMask	RO	16	16	Supported capabilities of this management class, bit set to 1 for affirmation of management support. Bits 0-7: are defined in 13.4.8.1 ClassPortInfo on page 734 Bits 8-15 for Boot Agent • Bit 8: **IsBtM_KeyNonVolatile** 1b = BtM_Key saved in non-volatile storage 0b = BtM_Key not saved in non-volatile storage • Bit 9-15: Reserved Class-specific bits, set to 0. Bits 8-15 for Boot Manager • Bit 8-13: Reserved Class-specific bits, set to 0. • Bit 14: **GracefulFailover** - This bit indicates if the Boot Manager shares subscription information with standby mangers. A value of 1 indicates that the DA retains subscriptions across fail-overs so a client does not have to re-subscribe if fail-over is successful. Requirements for setting this bit are specified in sections A5.6.7.3.1 Subscription Integrity on page 1348. • Bit 15: **IsContextPersistent** - This bit indicates if the manager persistently stores subscription information such that subscriptions are retained across reset, restarts, and power cycles. Requirements for setting this bit are specified in A5.6.7.3.1 Subscription Integrity on page 1348. See 13.4.8.1 ClassPortInfo on page 734 for more information on the CapabilityMask.

CA5-17: The BtA shall implement ClassPortInfo for the BootMgt class as described in 13.4.8.1 ClassPortInfo on page 734 and further specified in Table 367, “Boot Management ClassPortInfo:CapabilityMask,” on page 1295.

A5.6.2 BTM_KEYINFO

Table 368 BtM_KeyInfo Attribute

Component	Access	Offset (bits)	Length (bits)	Description
BtM_Key	R/W	0	64	The 8-byte Boot Management key used in all Boot Management MADs by all valid BootManagers. A BtM_KeyInfo:BtM_Key value of 0 means no BtM_Key check is performed by the BtA (see Table 369, “BtM_Key Check,” on page 1297).

Table 368 BtM_KeyInfo Attribute (Continued)

Component	Access	Offset (bits)	Length (bits)	Description
BtM_KeyProtectBits	R/W	64	2	See A5.6.2.3 “BtM_Key Operations” on page 1297 for details.
Reserved	R/W	66	14	Reserved
BtM_KeyLeasePeriod	R/W	80	16	Timer value used to indicate how long the BtM_KeyProtectBits are to remain non zero after a failed BtM_Key check. The value of the timer indicates the number of seconds for the lease period. With a 16 bit counter, the period can range from one second to approximately 18 hours. 0 shall mean infinite. See A5.6.2.5 “BtM_Key Recovery” on page 1299 for details.
BtM_KeyViolations	R/W	96	16	Number of MADs that have been received at this booting platform since power-on or reset that have been dropped due to a failed BtM_Key check. Counts the number of Boot Management MADs that have been received by the platform that have had an invalid BtM_Key. The counter Increments until the count reaches all 1s and then must be set back to zero to re-enable incrementing. When the BootManager sets this component to 0x0000 the counter is reset to 0x0000 and counting resumes. Setting the counter to a value other than zero results the counter being left unchanged.
Reserved	RO	112	1424	Reserved

A5.6.2.1 BTM_KEY GENERAL USE

The Boot Management Key (BtM_Key) provides a separate level of authentication that helps protect against receipt of bad management request messages. The BootManager includes the BtM_Key in the BootMgt MAD to obtain authorization. The BtM_Key is used to authenticate a trusted source. Similar to the model used for the M_Key and B_Key, this model assumes that the fabric has some level of physical security. While the BtM_Key is located in the header of the BootMgt MAD, BtM_Key handling depends on whether BootMgt MAD contains a request or a response message or is a BootMgt Trap.

The BootManager sets the BtM_Key using BootMgtSet(BtM_KeyInfo). A BootMgtGetResp shall return a MADHeader:BtM_Key of zero as specified in Table 369, “BtM_Key Check,” on page 1297. The Method, BtM_Key, and BtM_KeyProtectBits determine how the BtA handles the BtM_Key. If a key check fails, then the BtA silently drops the packet and increments the BtM_KeyViolations counter. If the BtA supports traps or notices, then it issues a Key Violation notice.

CA5-18: The BtM_KeyInfo:BtM_KeyViolations component shall be incremented once, each time the BtA receives a MAD for which the BtM_Key

check was performed according to Table 369, "BtM_Key Check," on page 1297 and failed. However, the counter shall not be incremented if its value is all 1's.

CA5-19: The BtM_KeyInfo:BtM_KeyViolations component shall be set to 0x0000 when a Set(BtM_KeyInfo) is received with BtM_KeyViolations=0x0000. A BtM_KeyViolations value other than 0x0000 shall leave the counter unchanged.

A5.6.2.2 BTM_KEY ASSUMPTIONS

Assumptions for using the BtM_Key are:

1) To use the correct key for each booting platform, the BootManager or a higher-level BtM_Key manager keeps track of the keys for the platforms that it is managing.
2) If a backup BootManager exists, it shares the BtM_Key for ease of fail-over.
3) The BootManager sets the BtM_Key, the BtM_KeyProtectBits, and the BtM_Key lease period via the BtM_KeyInfo Attribute with one BootMgtSet(BtM_KeyInfo) MAD. A successful completion of this assignment indicates to the BootManager that it has taken ownership of the booting platform.

A5.6.2.3 BTM_KEY OPERATIONS

The success and affect of the BtM_Key validation check depends on the value of the BtM_Key, BtM_KeyProtectBits, and on the method and attribute contained in the incoming MAD. If the key check succeeds then the BtA responds to the MAD. If the key check fails the MAD is silently dropped.

Table 369 BtM_Key Check

BtM_KeyInfo Component Values		Description
BtM_Key	**BtM_Key ProtectBits**	
zero	any	The BtM_Key contained in the MADHeader:BtM_Key of the MAD shall not be checked when the BtM_KeyInfo:BtM_Key is zero. As a result, no authentication is performed.

Table 369 BtM_Key Check (Continued)

BtM_KeyInfo Component Values		Description
BtM_Key	**BtM_Key ProtectBits**	
non-zero	00b	• BootMgtGet(*) and BootMgtTrapRepress(*) shall succeed for any key in the MADHeader:BtM_Key • BootMgtGet(BtM_KeyInfo) shall return the platform's BtM_Key in BootMgtGetResp(BtM_KeyInfo) allowing any BootManager to learn the BtM_Key of the port. • BootMgtSet(*) shall fail if MADHeader:BtM_Key does not match the BtM_KeyInfo:BtM_Key component of the platform.
non-zero	01b	• Any Boot Management MAD received at the port shall succeed if MADHeader:BtM_Key matches the BtM_KeyInfo:BtM_Key component of the platform. • BootMgtTrapRepress(*) shall succeed for any key in the MADHeader:BtM_Key • BtM_Key check on BootMgtGet(*) shall succeed for any key in the MADHeader:BtM_Key • BootMgtGetResp(BtM_KeyInfo) shall return the BtM_Key set to zero if MADHeader:BtM_Key does not match the BtM_KeyInfo:BtM_Key. This prevents the BootManager from learning the BtM_Key of the platform. • BootMgtSet(*) shall fail if MADHeader:BtM_Key does not match the BtM_KeyInfo:BtM_Key component of the platform.
non-zero	10b	• Any Boot Management MAD received at the port shall succeed if MADHeader:BtM_Key matches the BtM_KeyInfo:BtM_Key component of the platform • BootMgtTrapRepress(*) shall succeed for any key in the MADHeader:BtM_Key • BootMgtGet(*) and BootMgtSet(*) shall fail if MADHeader: BtM_Key does not match the BtM_KeyInfo:BtM_Key component of the platform.
non-zero	11b	Reserved

CA5-20: When the BtA receives a validated BootMgtGet() or BootMgtSet() and BtM_Key checking succeeds according to the rules specified in Table 8, "BtM_Key Check" on page 35, then the BtA shall generate a BootMgtGetResp().

CA5-21: The BtA shall perform the authentication determined by the contents of BtM_KeyInfo:BtM_Key and the BtM_KeyInfo:BtM_KeyProtectBits as per the behaviors described in Table 369, "BtM_Key Check," on page 1297.

CA5-22: If BtM_Key check specified in Table 369, "BtM_Key Check," on page 1297 fails, the BtA shall:

1) Silently drop the MAD.

2) Increment BtM_KeyViolations. Incrementing shall stop when the component reaches all 1s.
3) Send a Key Violation 0x0000 trap on all ports that have a non-zero ClassPortInfo:TrapLID if traps are supported by the BtA.
4) If Notice Queues are supported, a notice is posted into the Notice Queue.

CA5-23: The BootManager shall not check the MADHeader:BtM_Key in any received Boot Management class MAD.

When the BtA sends a MAD to the BootManager there is no requirement for the BootManager to do any type of key authentication. For simplicity and consistency, the MADHeader:BtM_Key is set to zero for all MADs issued by the BtA.

CA5-24: The BtA shall set MADHeader:BtM_Key to zero in all Boot Management MADs that it sends.

A5.6.2.4 BTM_KEY INITIALIZATION

CA5-25: At power-up, reset, or reboot, the BtM_Key, BtM_KeyProtectBits, and BtM_KeyLeasePeriod shall be set to zero if NVRAM is not used; otherwise, they shall be set to the values stored in NVRAM.

Using a BootMgtSet(BtM_KeyInfo), the BootManager may assign the subsequent BtM_Key, BtM_KeyProtectBits, and BtM_KeyLeasePeriod. Note that the BootManager must ensure that the lease period allows ample time for the BootManager to sweep the subnet to prevent the lease from expiring.

A5.6.2.5 BTM_KEY RECOVERY

The BtM_Key lease period timer starts when a BootManager sends the BtA a Boot Management class MAD whose BtM_Key fails the BtM_Key check. At this time, the booting platform sends a Key Violation trap to the BootManager (if traps are supported and if the BootManager stored its information in the trap components of the ClassPortInfo attribute). This trap serves as a request to the BootManager to refresh the lease period by issuing any MAD with a MADHeader:BtM_Key that matches the BtM_KeyInfo:BtM_Key. The lease period timer is reset to the value contained in BtM_KeyInfo:BtM_KeyLeasePeriod when any MAD is received with MADHeader:BtM_Key that matches the BtM_KeyInfo:BtM_Key.

If the BootManager fails to send a MAD with the BtM_Key that matches BtM_KeyInfo:BtM_Key, then the lease period expires - clearing the BtM_KeyProtectBits to zero and allowing anyone to read (and then set) the BtM_Key.

In the case when a booting platform initializes using NVRAM (e.g. **IsBtM_KeyNonVolatile**=1b) then BtM_Key, BtM_KeyProtectBits, BtM_KeyLeasePeriod are set to the values in local-persistent storage. The TrapLID is reset to zero and waits for the BootManager to come around to set the TrapLid. If the port receives a MAD that fails the BtM_Key check and the TrapLid is reset then the booting platform can not determine the LID of the BootManager needed to send the trap. In this case, the booting platform does not send the trap and the lease period timer can expire, causing eventual take over by a new BootManager.

With the BootMgtGet(BtM_KeyInfo), any BootManager can detect whether the BtM_Key is set (although hidden) based on the BtM_KeyProtectBits. If the BtM_KeyProtectBits are non-zero, the BtM_Key is set and hidden. Otherwise, the returned BtM_Key is the real one even if it is zero. Failure to get a response after some number of attempts is an indication that the BtM_Key is set and BtM_KeyProtectBits = 10b.

A5.6.2.6 LEASE PERIOD

A Lease Period is specified by setting the contents of the BtM_KeyInfo:BtM_KeyLeasePeriod component. It is intended to allow a BtM_Key to 'expire' if the BootManager inadvertently goes away without sharing the BtM_Key with backup BootManagers.

CA5-26: The lease period timer shall start counting down toward zero when a Boot Management MAD is received for which the BtM_Key check was performed according to Table 369, “BtM_Key Check,” on page 1297 and failed. If the lease timer count is already underway, it shall not be interrupted by the arrival of that MAD.

Furthermore, if a port is capable of sending Boot Management traps, a Key Violation trap described in Table 380, “Notice Details for Trap 0x0000 - KeyViolation,” on page 1341 is sent to the BootManager indicating that the lease timer has started counting. In response to that trap, the BootManager may refresh the Lease Period. If a BootManager with the proper BtM_Key has gone away, the Lease Period may expire.

CA5-27: The lease period timer shall cease counting down and shall be reset to the value contained in BtM_KeyInfo:BtM_KeyLeasePeriod component when any MAD is received with MADHeader:BtM_Key that matches the BtM_KeyInfo:BtM_Key.

CA5-28: The BtA shall set its BtM_KeyInfo:BtM_KeyProtectBits to zero when its lease period counter expires.

When the lease period expires, clearing the BtM_KeyProtectBits allow any BootManager to read (and then set) the BtM_Key.

CA5-29: When the BtM_KeyInfo:BtM_KeyLeasePeriod is set to zero, the lease period timer shall never expire.

Whether there is an out-of-band mechanism to reset data protected with a lease period of zero is outside the scope of the specification.

A5.6.3 PLATFORMBOOTINFO ATTRIBUTE

The BootManager informs the BtA of the desired behavior of the platform in order to assist in setting the booting policy of the subnet. The scope of this attribute is the booting platform. Table 370 "PlatformBootInfo Attribute" on page 1302 describes the format of the PlatformBootInfo attribute. The BootManager reads the booting platforms capabilities and current status by sending a BootMgtGet(PlatformBootInfo). This attribute also allows the BootManager to Set(PlatformBootInfo) components and therefore control the booting behavior of the platform.

Table 370 PlatformBootInfo Attribute

Component	Access	Offset (bits)	Length	Description
Capability	RO	0	32 bits	This component reports the capability of the Booting Platform 0b indicates booting platform does not support the capability. 1b indicates booting platform supports the capability. • **bit 0 - Extended Boot Environment** - Booting platform is able to load code from a ROM Repository that extends the boot environment and/or BtA capability. • **bit 1 - Proprietary Driver Load** - The booting platform is able to load proprietary device drivers from a ROM Repository • **bit 2 - IB Console Protocol** - The boot environment supports this protocol • **bit 3 - Proprietary Console Protocol** - The boot environment supports a proprietary console protocol. • **bit 4 - SRP Storage Protocol** - The boot environment supports this protocol • **bit 5 - Proprietary Storage Protocol** - The boot environment supports a proprietary storage protocol • **bit 6 - Network Boot-IB Network Model** - The boot environment supports a proprietary network boot protocol (without LAN NIC) see A5.1.6 Network Boot Method on page 1278. • **bit 7 - Network Boot- NIC Model** - The boot environment supports a proprietary network boot protocol (with LAN NIC) see A5.1.6 Network Boot Method on page 1278. • **bit 8 - BIS -** Booting platform can boot from Locator Records provided by a BIS. • **bit 9 - Persistent Boot** - Booting platform can boot from Locator Record information located in local non-volatile storage. • **bit 10 - Update ROM Locator Records** - BtA can write persistent ROM Repository information located in local non-volatile storage • **bit 11 - Update Console Locator Records** - BtA can write persistent console information located in local non-volatile storage • **bit 12 - Update OS Locator Records** - BtA can write persistent OS locator information located in local non-volatile storage • **bit 13 -** Reserved • **bit 14 - Immediate Reboot.** This booting platform has the capability of initiating an Immediate reboot to the platform when the BtA receives a BootMgtSet(NodeReboot). • **bit 15 - Graceful Reboot**. This booting platform has the capability of initiating a Graceful reboot of the platform when the BtA receives a BootMgtSet(NodeReboot). • **bit 16-31 -** Reserved
reserved	RO	32	4-bits	reserved

Table 370 PlatformBootInfo Attribute (Continued)

Component	Access	Offset (bits)	Length	Description
ROMRepositoryLocatorSource	R/W	36	4-bits	This component indicates how and in what order the boot platform locates a ROM Repository to extend its boot environment. When Capability(0) = 0b and Capability(1) = 0b, then the only valid value is 0xF. • **0x0** - BIS Only • **0x1** - BIS then Persistent • **0x2** - Persistent Only • **0x3** - Persistent then BIS • **0xF** - Disable/No ROM Repository for expansion.
ConsoleLocatorSource	R/W	40	4-bits	This component indicates how and in what order the boot platform locates a Console IOC or Server. When both Capability(2) = 0b and Capability(3) = 0b, then the only valid value is 0xF. • **0x0** - BIS Only • **0x1** - BIS then Persistent • **0x2** - Persistent Only • **0x3** - Persistent then BIS • **0xF** - Disable/No console
OsLocatorSource	R/W	44	4-bits	This component indicates how and in what order the boot platform locates an IOC and device containing its operating system loader for either storage or network booting. • **0x0** - BIS Only • **0x1** - BIS then Persistent • **0x2** - Persistent Only • **0x3** - Persistent then BIS • **0xF** - Disable/No OS boot loader
reserved	RO	48	8 bits	reserved
reserved	RO	56	6-bits	reserved
PlatformBootInfoSource	R/W	62	1-bit	This component indicates if the booting platform will query BIS for PlatformBootInfo - A booting platform may obtain PlatformBootInfo from either a BIS of from persistent information saved locally on the platform. 0b - The booting platform will query the BIS for PlatformBootInfo at the beginning of the boot process. This is the Default value 1b - The booting platform will obtain PlatformBootInfo from Persistent storage.
PortBootInfoSource	R/W	63	1-bit	This component indicates if the booting platform will query BIS for PortBootInfo - A booting platform may obtain PortBootInfo from either a BIS of from persistent information saved on the platform. 0b - The booting platform will query the BIS for PortBootInfo at the beginning of the boot process. This is the Default value 1b - The booting platform will obtain PortBootInfo from Persistent storage.
RomLocatorCount	RO	64	8 bits	This component specifies the number of ROM Repository Locator Records the booting platform can persistently save that point to a ROM Repository.

Table 370 PlatformBootInfo Attribute (Continued)

Component	Access	Offset (bits)	Length	Description
ConsoleLocatorCount	RO	72	8 bits	This component specifies the number of Console Locator Records the booting platform can persistently save that point to a console device/service.
OsLocatorCount	RO	80	8 bits	This component specifies the number of OS Locator Records the booting platform can persistently save that point to a OS boot loader located on a device or provided as a service that points to a device or service that can access the OS boot loader.
reserved	RO	88	8 bits	reserved
DeletePersisten-tRecords	RW	96	8 bits	This component instructs the BtA to erase persistent Locator Records bit 0 - 1b erases all Persistent RomRepositoryLocatorRecords from use bit 1 - 1b erases all Persistent ConsoleLocatorRecords from use bit 2 - 1b erases all Persistent OsLocatorRecords from use bit 3-7 - Reserved and set to zero This component is set to zero in BootMgtGetResp(PlatformBootInfo)
reserved	RO	104	24 bits	reserved

Table 370 PlatformBootInfo Attribute (Continued)

Component	Access	Offset (bits)	Length	Description
ExtendedBootProgress	RO	128	16 bits	This component reports the status of a booting platform while attempting to load the extended boot environment. The ROM Repository is accessed when loading the extended boot environment. **bit 0 - Not Attempted** - Platform has not attempted to use the ROM Repository to extend the boot environment. **bit 1 - In Progress** - Platform will use the ROM Repository. This status bit is set when the platform either accesses the Persistent ROM Repository Locator Record or queries the SA for a path to the BIS in order to locate a ROM Repository. **bit 2 - No ROM Repository Locator Record** - Platform either does not have a ROM Repository Locator Record or one or more RomRepositoryLocatorRecords are invalid. Both local-persistent and BIS Locator Records, if supported, are include in this status bit. **bit 3 - No Path Returned to ROM Repository (IOU not found**)- Platform cannot locate an IOU specified by a RomRepositoryLocatorRecord **bit 4 - ROM Repository not found on IOU** - The IOU in the RomRepositoryLocatorRecord does not contain a ROM Repository. **bit 5 - Cannot connect to ROM Repository** - After repeated attempts, the platform cannot make an RC connection to the ROM Repository. **bit 6 - ROM Repository image for Extension not found** - A ROM Repository does not contain an Extended Boot Image suitable for the platform. **bit 7:9** - Reserved and set to zeros **bit 10 - ROM Repository protocol error** - A protocol error was detected while connected to the ROM Repository causing the connection to be terminated. **bit 11** - Reserved and set to zeros. **bit 12** - Reserved and set to zeros. **bit 13 - ROM Repository image invalid, corrupted or incompatible** - A ROM Repository is corrupted or contains a corrupt image. **bit 14 - Extension process complete and operational** - The platform has successfully loaded the extended boot environment code. This status may be present even though other failures were detected. This can occur when multiple ROM Repositories are available. **bit 15** - Reserved and set to zeros **0x0000 - Disabled or Not Supported** - This platform does not report ExtendedBootProgress. **0x0001 - Set to this value at reset** (Not Attempted)
Reserved	RO	144	8 bits	Reserved

Table 370 PlatformBootInfo Attribute (Continued)

Component	Access	Offset (bits)	Length	Description
ExtendedBootState	RO	152	8 bits	This component reports the state of a booting platform attempting to load the extended boot environment. The ROM Repository is accessed when loading the extended boot environment. This component reports the success or failure of the platform to extend its boot environment. **0x00 - Set to this value at reset** **0x01 - In Progress** - Platform has begun to load the extended boot environment and has not detected a failure. **0x10 - Extended Boot Environment Failure** - The booting platform has failed to boot the OS because the boot environment could not be extended. **0x11 - Extended Boot Environment Failure** - The booting platform has failed to load the desired extended boot environment. The non-extended boot environment is/was used to boot the OS. **0xFE -Success** - The platform is using the extended boot environment **0xFF - Disabled or Not Supported** - This platform does not report ExtendedBootState. All other code points reserved

Table 370 PlatformBootInfo Attribute (Continued)

Component	Access	Offset (bits)	Length	Description
ConsoleBootProgress	RO	160	16 bits	This component reports Console Boot Progress - The Console device or service is normally enabled as early as possible in the boot process. Extending the boot environment and loading Console device drivers may be a prerequisite for making a console connection. **0x0001 - Set to this value at reset (Not Attempted)** **bit 0 - Not Attempted** - Platform has not yet attempted to use the Console **bit 1 - In Progress** - Platform will use a Console device or service. This status bit is set when the platform either accesses the local-persistent ConsoleLocatorRecord or queries the SA for a path to the BIS in order to locate a console. **bit 2 - No ConsoleLocatorRecord** - Platform either does not have a ConsoleLocatorRecord or one or more ConsoleLocatorRecords are invalid. Both Local-Persistent and BIS Locator Records, if supported are include in this status bit. **bit 3 - No Path Returned for Console** (IOU or service not found)- Platform cannot access one or more Console targets because a Path Record was not returned from the SA. **bit 4 - Console IOU or Server not found** - The IOU or server identified by the ConsoleLocatorRecord is not responding to MADs. **bit 5 - Console GUID/SID not found** - The GUID/SID in the ConsoleLocatorRecord cannot be found. **bit 6 - Cannot connect to Console** - After repeated attempts, the platform cannot make an RC connection to the Console. **bit 7** - Reserved and set to zero **bit 8 - Console ProtocolName not found** - ProtocolName not found in AdditionalInfo when ConsoleLocatorRecord:Protocol(7)=1b **bit 9 - Console protocol not supported** - The platform does not support the protocol used by the device or service. **bit 10 - Console protocol error** - A protocol error was detected while connected to the Console causing the connection to be terminated. **bit 11 - Console Device Driver not found** - The platform cannot locate a Console Device Driver suitable for this platform. **bit 12** - Reserved and set to zeros **bit 13 - Console Device Driver image invalid, corrupted or incompatible** - The console device driver found in the ROM Repository is not usable. **bit 14 - Console operational** - The platform has successfully connected to the console device or service. **bit 15** - Reserved and set to zeros **0x0000 - Disabled or Not Supported** - This platform does not report Console Informational Status. **0x0001 - Set to this value at reset (in Progress**)
Reserved	RO	176	8 bits	Reserved

Table 370 PlatformBootInfo Attribute (Continued)

Component	Access	Offset (bits)	Length	Description
ConsoleBootState	RO	184	8 bits	This component reports Console Boot State - This component reports the success or failure of the platform to use the console. **0x00 - Set to this value at reset** **0x01 - In Progress** - Platform has begun to access the IB console and has not detected a failure. **0x10 - Console Device Driver Load Failure** - The platform could not locate a Device Driver for the console device or service - the platform will attempt to load the OS. **0x11 - Boot Failure Console Device Driver** - The platform has failed to boot the OS because the platform could not locate a Device Driver for the console device or service. **0x20 - Console Failure** - The platform has located a DD but failed to connect to a console - the platform will attempt to load the OS. **0x21 - Boot Failure Console** - The platform has located a DD but has failed to connect to a console - the platform will not attempt to load the OS. **0xFE -Success** - The platform is using the console **0xFF - Disabled or Not Supported** - This platform does not report ConsoleBootState. **All other code points reserved**

Table 370 PlatformBootInfo Attribute (Continued)

Component	Access	Offset (bits)	Length	Description
OSBootProgress	RO	192	16 bits	This component reports the status of the platform's OS boot process. **0x0001 - Set to this value at reset (Not Attempted)** **bit 0 - Not Attempted** - Platform has not yet attempted to load the OS loader from the boot device or server. **bit 1 - In Progress** - This status is set when the platform either accesses the Local-Persistent OSLocatorRecords or queries the SA for a path to the BIS in order to locate the boot device. **bit 2 - No OSLocatorRecord** - Platform either does not have an OS Locator Record or one or more OS Locator Records are invalid. Both Local-Persistent and BIS Locator Records, if supported are include in this status bit. **bit 3 - No Path Returned for Device/Service** (IOU or service not found)- Platform cannot access one or more boot targets because a Path Record was not returned from the SA. **bit 4 - OS Boot Device or Server not found** - The device or server identified by an OsLocatorRecord cannot be found. **bit 5 - IocGUID/SID not found** - The IocGUID or SID in the OsLocatorRecord cannot be found. **bit 6 - Cannot connect to OS Boot Target** - After repeated attempts, the platform cannot make a connection to the Boot Target. **bit 7 -** Reserved and set to zero **bit 8 - I/O ProtocolName not found** - ProtocolName not found in AdditionalInfo when OsLocatorRecord:Protocol(7)=1b **bit 9 - OS Device protocol Unknown** - The platform does not support the protocol used by the device. **bit 10 - OS Boot protocol error** - A protocol error was detected while connected to the Boot target causing the connection to be terminated. **bit 11 - OS Boot Device Driver not found** - The platform cannot locate an OS Boot Device Driver suitable for this platform. **bit 12 - OS Boot AdditionalInfo** - Device in AdditionalInfo not found or AdditionalInfo invalid. **bit 13 - OS Boot Device Driver image invalid, corrupted or incompatible** - The console device driver found in the ROM Repository is corrupt. **bit 14 - OS Boot Loader operational** - The platform has successfully loaded the OS Boot Loader. **bit 15** - Reserved and set to zeros **0xFFFF - Disabled or Not Supported** - This platform does not report OSBootProgress. **0x0001 - Set to this value at reset (in Progress**)
Reserved	RO	208	8 bits	Reserved

Table 370 PlatformBootInfo Attribute (Continued)

Component	Access	Offset (bits)	Length	Description
OSBootState	RO	216	8 bits	This component reports the state of the platform in booting its OS. It reports the success or failure to locate an OS boot loader. **0x00 - Set to this value at reset** **0x01 - In Progress** - Platform has begun to access the OS boot loader and has not detected a failure. **0x00 - Not Urgent** - Platform has not detected an urgent error accessing the OS Loader. Set to this value at reset (in Progress) **0x11 - OS Boot Failure Device Driver** - The platform has failed to boot the OS because the platform could not locate a Device Driver for any of the boot devices. **0x21 - OS Boot Failure, no Boot Loader** - The platform has failed to locate any boot loader (i.e., IOU/IOC not present, no boot device, or no boot loader on boot device, etc.). **0xFE -Success** - The platform is using the boot device. **0xFF - Disabled or Not Supported** - This platform does not report OSBootState. **All other code points reserved**
BootPlatformUUID	RO	224	128 bits	A 128-bit universally unique identifier (UUID) as defined by ISO/IEC 11578 that uniquely identifies the booting platform.
PlatformInfo	RO	352	1024 bits (128-Bytes)	A TLV encoded component containing multiple packed elements as described in A5.6.3.2 "PlatformInfo" on page 1311.
Reserved	RO	1376	160	Reserved

CA5-30: A BtA shall report its Capability via the PlatformBootInfo attribute returned in the BootMgtGetResp method. See Capability in Table 370 "PlatformBootInfo Attribute" on page 1302.

CA5-31: If PlatformBootInfo:Capability(10:12) is non-zero then the BtA shall persistently save the R/W components of PlatformBootInfo and Port-BootInfo in non-volatile storage.

CA5-32: If the BtA does not support BootMgtSet(PlatformBootInfo) or BootMgtSet(PortBootInfo) then the BtA shall return MAD:Status(2:4) = 0x3 in the response of the request.

oA5-8: If the booting platform attempts to locate a ROM repository using Local-Persistent ROMRepositoryLocatorRecords, it shall use them in the order specified by their AttributeModifier.

oA5-9: If the booting platform attempts to locate a Console using Local-Persistent ConsoleLocatorRecords, it shall use them in the order specified by their AttributeModifier.

oA5-10: If the booting platform attempts to locate an OS boot loader using Local-Persistent OsLocatorRecords, it shall use them in the order specified by their AttributeModifier.

oA5-11: A BtA shall report the number of RomRepositoryLocatorRecords it can save in non-volatile storage by returning the count of locator records in BootMgtGetResp(PlatformBootInfo:RomLocatorCount).

oA5-12: A BtA shall report the number of ConsoleLocatorRecords it can save in non-volatile storage by returning the count of locator records in BootMgtGetResp(PlatformBootInfo:ConsoleLocatorCount).

oA5-13: A BtA shall report the number of OsLocatorRecords it can save in non-volatile storage by returning the count of locator records in BootMgtGetResp(PlatformBootInfo:OsLocatorCount).

A5.6.3.1 BOOTPLATFORMUUID

The BtA uniquely identifies a booting platform by its BootPlatformUUID. The 128-bit UUID was selected over a 64-bit GUID (EUI-64) because it can be generated easily by software, which de-couples the boot environment from any particular hardware. For example, it removes the necessity for a hardware serial number to be machine readable. Software generation is also advantageous for retrofit and field upgrades in machines that are not specifically designed for InfiniBand. A BootManager does not need to interpret the BootPlatformUUID, but rather uses it as an opaque value to match with locator records assigned to that BootPlatformUUID.

CA5-33: The booting platform shall provide a persistent UUID in PlatformBootInfo:BootPlatformUUID that uniquely identifies the booting platform.

oA5-14: If a booting platform supports BIS, then the booting platform shall provide a persistent BootPlatformUUID in the BisQuery() that uniquely identifies the booting platform. The UUID in the BIS query shall be the same value reported in the BtMGetResp(PlatformBootInfo).

A5.6.3.2 PLATFORMINFO

The BootPlatformUUID is a component that a BootManager uses to identify a booting platform. The PlatformInfo component is intended as supplemental information to identify those characteristics of a booting platform that may influence the set of locator records selected for the booting platform.

Elements in PlatformInfo are in TLV format. The first byte is the Type, the second byte is the Length (number of bytes in the value string), and the remainder of the element is the value string, in UTF-8 format. Type codes are specified in Table 371. The PlatformInfo component contains a variable number of variable length elements and is terminated with the null value (0x00). All bytes after the termination byte should also be null bytes.

Each respective booting platform vendor is responsible for defining its own value string definition. For instance the Platform vendor defines unique values for each of its products and firmware vendor defines unique values for each of its products. An element with a Length of zero means that the booting platform does not know the value for that element.

The recommended practice is for the booting platform to order elements by their Type value, lowest to highest.

Table 371 PlatformInfo Elements

Type Value	Length Value	Description (All strings are in UTF-8 and the content of each element is vendor specific)
0x00	0x00	Marks end of elements - ignore remainder of component data following this Type code
0x01	variable	**Platform Vendor Name String** This string provides the name of the vendor that manufactured the booting platform.
0x02	variable	**Platform Vendor Model/Type String** This string provides the model and type of the booting platform.
0x03	variable	**Platform Serial Number String** This string provides the serial number of the booting platform.
0x04	variable	**Firmware Vendor Name String** This string provides the name of the firmware vendor that manufactured the boot environment code (e.g., BIOS vendor.).
0x05	variable	**Firmware Version String** This string provides the version of the firmware code.
0x06	variable	**CPU Vendor Name String** This string provides the name of the CPU vendor
0x07	variable	**CPU Vendor Version String** This string provides the CPU version, stepping, etc.
0x08	variable	**Platform Name String** This string provides the local name assigned the booting platform - usually assigned by the System Administrator to identify the platform by name.
0x09	variable	**OS Name String** This string provides the name of the preferred Operating System.

A5.6.3.3 BOOTING PLATFORM CAPABILITY

Platform vendors provide BtAs with varying degrees of functionality. The booting platform's Capability component in PlatformBootInfo reports the functionality of the BtA to the BootManager. See Table 370.

A5.6.3.3.1 EXTENDED BOOT ENVIRONMENT

A 1b returned in PlatformBootInfo:Capability(0)) indicates that the boot environment can be expanded by loading additional code from a ROM Repository.

Each vendor that desires to expand its boot environment using the ROM Repository should test the repository Image Descriptor (see A5.12.8 "IMAGE Descriptor" on page 1375). It is expected that each platform vendor checks the ImageAuthority component in the Image Descriptor for an ImageAuthority and ImageType recognized by the boot environment of the booting platform. Other checks such as validating the content of the image may also be necessary. For more information on using the ROM Repository see A5.12 "ROM Repository" on page 1365.

A5.6.3.3.2 PROPRIETARY DRIVER LOAD

Capability(1)) indicates that the booting platform has the capability of loading proprietary drivers. Proprietary Driver Load is a method or process to load Storage, Network, and/or Console device drivers from another node (e.g the ROM Repository) into the booting platform. This allows the I/O subsystem to deliver the boot platform code (3rd party drivers) enabling devices, services, and protocols present within the IB fabric. This firmware is typically dependent on the HW-platform (BIOS, IEEE1275, EFI, etc.) when loading code for the boot environment and also OS dependant when loading code for the OS (i.e. the driver is specific to the OS). When the booting platform is booting from a device, the platform may search for the ROM Repository on the same IOU that contains the IOC, or may use a centralized ROM Repository[35]. If the booting platform is booting from a boot service or if the correct device driver is not found, then the platform should search the ROM repositories pointed to by the RomRepositoryLocatorRecord(s).

A5.6.3.3.3 PROTOCOLS SUPPORTED

Capability(0:7) indicates that the booting platform supports particular protocols.

- Capability(0) or Capability(1) indicates that the booting platform supports the IB ROM Repository protocol described in Table 370 "PlatformBootInfo Attribute" on page 1302.

35. Console and OS Locator Records contains a component (DeviceDriverLocation) that informs the booting platform of which repository to search first.

- Capability(2) indicates that the booting platform supports the IB Console protocol described in Annex A2: Console Service Protocol on page 1140.
- Capability(3) indicates that the booting platform supports a proprietary Console protocol the details of which are outside the scope of IBA.
- Capability(4) indicates that the booting platform supports the SRP Storage protocol described in Annex A1: I/O Infrastructure on page 1121.
- Capability(5) indicates that the booting platform supports a proprietary Storage protocol the details of which are outside the scope of IBA.
- Capability(6) indicates that the booting platform supports a Network protocol that uses a LAN NIC.
- Capability(7) indicates that the booting platform supports a Network protocol that does not use a LAN NIC. The details of which are outside the scope of IBA (see Section 5.1.6, "Network Boot Method," on page 1278).

A5.6.3.3.4 BOOT RESOLUTION METHODS SUPPORTED

There are two sources for locator records, Local-Persistent storage or BIS. Local-Persistent Locator Records are saved in non-volatile storage on the booting platform. Local-Persistent Locator Records can be set using Boot Management attributes. BIS records are provided by a BIS service when the service is queried by the booting platform.

- Capability(8) indicates that the boot platform can query the BIS using BIS class MADs and use the BIS Locator Records returned for locating a ROM Repository, a console and a boot device.
- Capability(9) indicates that the booting platform can use Local-Persistent Locator Records located locally in non-volatile storage.

CA5-34: A BtA that supports the Local-Persistent Locator Records (Capability(9)) shall have the capability to read all RomRepositoryLocatorRecords, ConsoleLocatorRecords, and OSLocatorRecords in non-volatile storage.

oA5-15: A booting platform whose BtA sets Capability(8) shall have the capability to query the BIS for RomRepositoryLocatorRecord, ConsoleLocatorRecords, OSLocatorRecords, PlatformBootInfo, and PortBootInfo attributes.

A5.6.3.3.5 UPDATE LOCATOR RECORDS SUPPORTED

Capability(10) indicates that the BtA has write access to Local-Persistent Locator Records pointing to the ROM Repository.

oA5-16: A BtA that supports Update ROM Locator Records, Capability(10) = 1b, shall have the capability to save RomLocatorCount RomRepositoryLocatorRecords in non-volatile storage across power cycles and RomLocatorCount shall be greater than zero. A 0b indicates that the booting platform does not have this capability.

Capability(11) indicates that the BtA has write access to Local-Persistent Locator Records pointing to the Console object.

oA5-17: A BtA that supports Update Console Locator Records, Capability(11) = 1b shall have the capability to save ConsoleLocatorCount ConsoleLocatorRecords in non-volatile storage across power cycles and ConsoleLocatorCount shall be greater than zero. A 0b indicates that the booting platform does not have this capability.

Capability(12) indicates that the BtA has write access to Local-Persistent Locator Records pointing to the OS boot loader.

oA5-18: A BtA that supports Update OS Boot Locator Records, Capability(12) = 1b shall have the capability to save OsLocatorCount OsLocatorRecords in non-volatile storage across power cycles and OsLocatorCount shall be greater than zero. A 0b indicates that the booting platform does not have this capability.

A5.6.3.4 BOOT RECORD LOCATOR SOURCES

The ROMRepositoryLocatorSource, ConsoleLocatorSource, and OsLocatorSource components inform the booting platform of the Locator Record sources (Local-Persistent or BIS) to use when booting. Each locator selects the boot resolution method used and the order in which the method shall be used. BIS can only be used if Capability(8) = 1b. Local-Persistent can only be used if Capability(9) = 1b. These attributes can be set by the BootManager or the booting platform can query the BIS (see BisQuery in Annex A6: Boot Information Service on page 1403).

The component encode determines if either BIS or Local-Persistent methods should be used and if both are used which is attempted first.

ROMRepositoryLocatorSource

When the ROMRepositoryLocatorSource contains a 0x0 or 0x1 the booting platform will query the BIS to determine the location of a ROM Repository when required to expand the boot environment or access Proprietary drivers. When the ROMRepositoryLocatorSource contains a 0x3 the booting platform will query the BIS only if necessary after using RomRepositoryLocatorRecords saved in non-volatile storage.

When the ROMRepositoryLocatorSource contains a 0x2 or 0x3 the booting platform will use the Local-Persistent RomRepositoryLocator-Records to determine the location of the Repository. When the ROMRepositoryLocatorSource contains a 0x1 the booting platform will use Local-Persistent RomRepositoryLocatorRecords only if necessary after querying the BIS.

ConsoleLocatorSource

When the ConsoleLocatorSource contains a 0x0 or 0x1 the booting platform will query the BIS to determine the location of the Console device or service. When the ConsoleLocatorSource contains a 0x3 the booting platform will query the BIS only if necessary after using Local-Persistent ConsoleLocatorRecords saved in non-volatile storage.

When the ConsoleLocatorSource contains a 0x2 or 0x3 the booting platform will use the Local-Persistent ConsoleLocatorRecords to determine the location of the Console device or service. When the ConsoleLocator-Source contains a 0x1 the booting platform will use Local-Persistent ConsoleLocatorRecords only if necessary after querying the BIS.

OsLocatorSource

The OsLocatorSource is a way that a booting platform selects services and IOCs. Specifically, a mechanism by which a booting platform can resolve or select a set of Locator Records that the booting platform uses to find a Boot Device or a Boot Service.

When the OsLocatorSource contains a 0x0 or 0x1 the booting platform will query the BIS for OsLocatorRecords. When the OsLocatorSource contains a 0x3 the booting platform will query the BIS only if necessary after using Local-Persistent OsLocatorRecords saved in non-volatile storage.

When the OsLocatorSource contains a 0x2 or 0x3 the booting platform will use its Local-Persistent OsLocatorRecords. When the OSLocatorSource contains a 0x1 the booting platform will use Local-Persistent OsLocator-Records only if necessary after querying the BIS.

oA5-19: If the boot platform supports RomRepositoryLocatorRecords, then the platform shall use PlatformBootInfo:RomRepositoryLocator-Source to determine the method to use to locate the ROM Repository.

oA5-20: If the boot platform supports ConsoleLocatorRecords, then the platform shall use PlatformBootInfo:ConsoleLocatorSource to determine the method to use to locate the Console.

oA5-21: If the boot platform supports OsLocatorRecords, then the platform shall use PlatformBootInfo:OsLocatorSource to determine the method to use to locate the OS loader.

oA5-22: The BtA shall set MAD:Status(2:4) to 0x3 in the response if a BootMgtSet(PlatformBootInfo) attempts to Set() a component to an unsupported value inconsistent with PlatformBootInfo:Capability.

oA5-23: If PlatformBootInfo:OsLocatorSource is set to 0xF then the platform shall not boot from the IB fabric.

A BootManager would set PlatformBootInfo:OsLocatorSource to 0xF while updating OsLocatorSource records so that if the platform re-booted while the Boot manager is updating OsLocatorRecords the OsLocatorRecords would not be used as a pointer to the OS boot loader.

oA5-24: If PlatformBootInfo:ConsoleLocatorSource is set to 0xF then the platform shall not access the console over the IB fabric.

oA5-25: If PlatformBootInfo:RomRepositoryLocatorSource is set to 0xF then the platform shall not use RomRepositoryLocatorRecord to locate a ROM Repository.

When a booting platform has multiple ports on multiple subnets or multiple partitions then multiple BootManagers may share the platform BtM_Key. To keep each BootManager informed of changes to Locator Records, a Trap is issued out all ports that have a non-zero ClassPortInfo:TrapLID- when any Locator Record is modified.

A5.6.3.5 RECORD COUNT

The BtA reports the number of Local-Persistent Locator Records capable of being persistently saved by the platform for each of the 3 types of Local-Persistent locator records, namely RomRepositoryLocatorRecord, ConsoleLocatorRecord, and OsLocatorRecord.

A non-zero count returned in RomLocatorCount, ConsoleLocatorCount, or OsLocatorCount indicates that the booting platform has the ability to persistently save count number of Local-Persistent Locator Records across power cycles.

oA5-26: BootMgtGetResp(PlatformBootInfo:RomLocatorCount) shall indicate the number of RomRepositoryLocatorRecords the platform can persistently store. A 0x00 indicates that the platform can not save any RomRepositoryLocatorRecords.

oA5-27: BootMgtGetResp(PlatformBootInfo:ConsoleLocatorCount) shall indicate the number of ConsoleLocatorRecords the platform can persistently store. A 0x00 indicates that the platform can not save any ConsoleLocatorRecords.

oA5-28: BootMgtGetResp(PlatformBootInfo:OsLocatorCount) shall indicate the number of OsLocatorRecords the platform can persistently store. A 0x00 indicates that the platform can not save any OsLocatorRecords.

oA5-29: A BtA shall set MAD:Status(2:4) to 0x3 in BootMgtGetResp() when it receives a Set(RomRepositoryLocatorRecord) and the AttributeModifier is equal to or greater than the number of RomRepositoryLocatorRecords indicated in PortBootInfo:RomLocatorCount.

oA5-30: A BtA shall set MAD:Status(2:4) to 0x3 in BootMgtGetResp() when it receives a Set(ConsoleLocatorRecord) and the AttributeModifier is equal to or greater than the number of ConsoleLocatorRecords indicated in PortBootInfo:ConsoleLocatorCount.

oA5-31: A BtA shall set MAD:Status(2:4) to 0x3 in BootMgtGetResp(GetResp) when it receives a Set(OsLocatorRecord) and the AttributeModifier is equal to or greater than the number of OsLocatorRecords indicated in PortBootInfo:OsLocatorCount.

The BootManager erases individual persistent locator records by issuing a BootMgtSet() specifying the attribute (RomRepositoryLocatorRecord, ConsoleLocatorRecord, or OsLocatorRecord), the AttributeModifier to indicate the slot, and setting the BootMgtData component to zero.

The BtA shall not use a LocatorRecord containing all zero's (i.e. skips this record when booting).

The BtA may report non-initialized, invalid, or otherwise unusable Local-Persistent locator records by setting the BootMgtData component to zero in the BootMgtGetResp() of a RomRepositoryLocatorRecord, ConsoleLocatorRecord, or OsLocatorRecord read.

CA5-35: The BtA shall return RomRepositoryLocatorRecord in the BootMgtGetResp() with BootMgtData of the MAD(64:256) equal to all zeros (see Figure 274 "Boot Management MAD" on page 1286) when the locator record specified by the AttributeModifier has never been initialized, has been erased, or is otherwise unusable.

CA5-36: The BtA shall return ConsoleLocatorRecord in the BootMgtGetResp() with BootMgtData of the MAD(64:256) equal to all zeros (see Figure 274 "Boot Management MAD" on page 1286) when the locator record specified by the AttributeModifier has never been initialized, has been erased, or is otherwise unusable.

CA5-37: The BtA shall return OsLocatorRecord in the BootMgtGetResp() with BootMgtData of the MAD(64:256) equal to all zeros (see Figure 274 "Boot Management MAD" on page 1286) when the locator record speci-

fied by the AttributeModifier has never been initialized, has been erased, or is otherwise unusable.

A5.6.3.6 DELETING PERSISTENT RECORDS

When a configuration change affects the booting policy the BootManager may need to erase Locator Records on a booting platform. This component allows the BootManager to erase all the Local-Persistent Locator Records on a booting platform or just a particular type.

oA5-32: If PlatformBootInfo:Capability(10) is 1b and the BtA receives a Set(PlatformBootInfo) with DeletePersistentRecords(0) set to 1b then the BtA shall erase all RomRepositoryLocatorRecord's from non-volatile storage.

oA5-33: If PlatformBootInfo:Capability(11) is 1b and the BtA receives a Set(PlatformBootInfo) with DeletePersistentRecords(1) set to 1b then the BtA shall erase all ConsoleLocatorRecord's from non-volatile storage.

oA5-34: . If PlatformBootInfo:Capability(12) is 1b and the BtA receives a Set(PlatformBootInfo) with DeletePersistentRecords(2) set to 1b then the BtA shall erase all OsLocatorRecord's from non-volatile storage.

A5.6.3.7 STATUS COMPONENTS

The Status components allows the BtA to report multi-step operations when extending the boot environment, making a console connection, or accessing the OS boot loader. There are 2 status categories, boot progress and boot state. The status components in PlatformBootInfo are especially useful when a booting platform does not support Traps.

The informational progress components (ExtendedBootProgress, ConsoleBootProgress, OSBootProgress) report failures while processing one or more Locator Records. Once a bit comes on, it stays on until the platform resets or reboots.

The BootState components (ExtendedBootState, ConsoleBootState, OSBootState) report severe failures in which the platform no longer attempts to use the ROM Repository, the Console, or load the OS. Once an error code is set it stays set until the platform resets.

ExtendedBootProgress - This status component allows the BtA to report the status of the platforms attempt to expand its boot environment. ExtendedBootProgress is set to 0x0001 at power on reset or reboot. If the platform will expand its boot environment, the BtA sets this ExtendedBootProgress(1) - In Progress. All other bits in ExtendedBootProgress are described in Table 370 "PlatformBootInfo Attribute" on page 1302.

ExtendedBootState - This status component allows the BtA to report the results of the platforms attempt to expand its boot environment using one or more Locator Records. The ExtendedBootState component is described in Table 370 "PlatformBootInfo Attribute" on page 1302.

ConsoleBootProgress - This status component allows the BtA to report the status of the platforms attempt make a connection to a console. ConsoleBootProgress is set to 0x0001 at power on reset or reboot. If the platform uses an IB console during boot, the BtA sets this ExtendedBootProgress(1) - In Progress. All other bits in ConsoleBootProgress are described in Table 370 "PlatformBootInfo Attribute" on page 1302.

ConsoleBootState - This status component allows the BtA to report the results of the platforms attempt to connect to a Console using one or more Locator Records. The ConsoleBootProgress component is described in Table 370 "PlatformBootInfo Attribute" on page 1302.

OSBootProgress - This status component allows the BtA to report the status of the platforms attempt load the OS loader from a boot device. OSBootProgress is set to 0x0001 at power on reset or reboot. If the platform uses an IB OSLocatorRecord during boot, the BtA sets this OSBootProgress(1) - In Progress. All other bits in ConsoleBootProgress are described in Table 370 "PlatformBootInfo Attribute" on page 1302.

OSBootState - This status component allows the BtA to report the results of the platforms attempt to load the OS loader onto the boot platform using one or more Locator Records. The OSBootState component is described in Table 370 "PlatformBootInfo Attribute" on page 1302.

A5.6.4 PORTBOOTINFO ATTRIBUTE

Components in this attribute have a port scope and indicate the relative priority of the port in locating a BIS, ROM Repository, Console, and an OS boot loader. In addition, three time-out components determine the minimum amount of time the booting platform should spend attempting to find or make a connection.

Table 372 describes the format of the PortBootInfo attribute. PortBootInfo attribute allows the BootManager to set the port time-out value and port priorities with respect to other ports on the same booting platform. Time-outs are used so the boot environment can determine the maximum amount of time it should take to make a connection to a device or service through the port and the minimum time the booting platform should continue to attempt making a connection.

Table 372 PortBootInfo Attribute

Component	Access	Offset (bits)	Length	Description
BISPortPriority	R/W	0	2-bits	This component indicates the priority of this port to locate the BIS managing locator records used for booting. • 11b is the highest priority • 10b is medium priority • 01b is the lowest and the default • 00b indicates that this port should not be used to query the BIS.
RomPortPriority	R/W	2	2-bits	This component indicates the priority of this port to locate the RomRepository using RomRepositoryLocatorRecords. • 11b is the highest priority • 10b is medium priority • 01b is the lowest and the default • 00b indicates that this port should not be used to locate the RomRepository.
ConsolePortPriority	R/W	4	2-bits	This component indicates the priority of this port to locate the Console using ConsoleLocatorRecords. • 11b is the highest priority • 10b is medium priority • 01b is the lowest and the default • 00b indicates that this port should not be used to locate the Console.
IocPortPriority	R/W	6	2-bits	This component indicates the priority of this port to locate an I/O unit containing the IOC specified by an OsLocatorRecord. • 11b is the highest priority • 10b is medium priority • 01b is the lowest and the default • 00b indicates that this port should not be used to locate OS boot loader

Table 372 PortBootInfo Attribute (Continued)

Component	Access	Offset (bits)	Length	Description
NetworkBootPortPriority	R/W	8	2-bits	This component indicates the priority of this port when attempting an IB network boot (see A5.1.6 on page 1278). • 11b is the highest priority • 10b is medium priority • 01b is the lowest and the default • 00b indicates that this port should not be used for IB Network booting
reserved	RO	10	22-bits	reserved
InitTimeout	R/W	32	16-bits	The amount of time (in 100 msec increments) from power on reset that the booting platform allows for subnet resources to become operational. The factory default is ~2 min. = 0x04b0. This allows sufficient time for the IOU or services to become active and to prevent this booting platform from timing out too early and moving on to the next Locator Record.
BisTimeout	R/W	48	16-bits	Specifies the maximum amount of time (100 mSec increments) from power on reset that a BIS takes to become operational. The factory default is ~2 min. = 0x04b0. This allows sufficient time for the BIS to become active and to prevent this booting platform from timing out too early and moving on to the next BIS.
EndNodeTimeout	R/W	64	16-bits	The period of time the booting platform allows an endnode to respond to the first MAD. The booting platform continues to retry the MAD until EndNodeTimeout has expired or the endnode become operational. Each endnode the booting platform sends a MAD to is allowed EndNodeTimeout period of time to respond to the MAD before the booting platform determines the endnode unreachable. The time-out is specified in 100 mSec increments. • 0x0000 = 0 mS - Endnodes require no additional time to become operational • 0x0001 = 100 mS = default • 0x0002-0xFFFF = Additional valid time-out values Once an endnode responds, the booting platform no longer has to retry waiting for the endnode to become operational.
Reserved	RO	80	182 Bytes	Reserved

CA5-38: Every port shall have an InitTimeout parameter whose factory default is 0x04b0.

InitTimeout may be configured locally by a system administrator and can be set by the BootManager when the platform contains a BtA that supports BootMgtSet(PortBootInfo).

oA5-35: If the booting platform supports BIS, then every port shall have a BisTimeout parameter whose factory default is 0x04b0.

BisTimeout may be configured locally by a system administrator and can be set by the BootManager when the platform contains a BtA that supports BootMgtSet(PortBootInfo).

oA5-36: If BootMgtSet(PortBootInfo) is supported, the BtA shall persistently save the values of all PortBootInfo R/W components in non-volatile storage.

oA5-37: If BootMgtSet(PortBootInfo) is supported, the booting platform shall use BootMgtSet(PortBootInfo) component values only on the port in which the MAD was received.

oA5-38: If the Platform supports BIS then the platform shall attempt to query the BIS for Locator Records through ports in their PortBootInfo:BISPortPriority order. If PortBootInfo:BISPortPriority = 0x0 then the booting platform shall not attempt to query the BIS through this port.

CA5-39: When the Platform uses RomRepositoryLocatorRecords, then the platform shall attempt to access the RomRepository through ports in their PortBootInfo:RomPortPriority order. If PortBootInfo:RomPortPriority = 0x0 then the booting platform shall not attempt to access the RomRepository through that port.

oA5-39: If the Platform supports the IB Console then the platform shall attempt to access the IB Console using Locator Records through ports in their PortBootInfo:ConsolePortPriority order. If PortBootInfo:ConsolePortPriority = 0x0 then the booting platform shall not attempt to access the IB Console through this port.

oA5-40: If the Platform supports booting over IB (e.g. Capability(4:7) is non-zero) then the platform shall attempt to access the OS boot loader, through an IOC using Locator Records through ports in their PortBootInfo:IocPortPriority order. If PortBootInfo:IocPortPriority = 0x0 then the booting platform shall not attempt to access the OS boot loader through this port.

oA5-41: If the platform supports Capability(6), IB Network Boot, then the booting platform shall attempt network booting through ports in their PortBootInfo:NetworkBootPortPriority order. The booting platform shall not attempt an IB Network Boot through a port that has a PortBootInfo:NetworkBootPortPriority of zero.

oA5-42: The booting platform shall allow at least InitTimeout amount of time for all subnet resources to become operational. The period of time

begins at the last power on reset and extends for InitTimeout period measured in 100 mSec increments.

oA5-43: The booting platform shall allow at least BisTimeout amount of time for a BIS to become operational. The period of time begins at the last power on reset and extends for BisTimeout period measured in 100 mSec increments.

oA5-44: The booting platform shall allow at least EndNodeTimeout amount of time for an endnode to become operational. The period of time begins when the 1st MAD is sent to the particular end node and extends for EndNodeTimeout period measured in 100 mSec increments.

A5.6.4.1 BISPORTPRIORITY

BIS Port priority notifies the BtA of which port it is more likely to find a BIS that can supply ROM Repository, Console, and OS Locator Records. This component is useful when a booting platform attaches to multiple subnets.

A5.6.4.2 ROMPORTPRIORITY

RomPortPriority identifies which ports are more likely to find the RomRepository. This component is useful when the booting platform attaches to multiple subnets. When the booting platform processes a RomRepositoryLocatorRecord then the booting platform should access the RomRepository through the port with the highest priority first. If that fails, the booting platform proceeds with lower priority ports in priority order. When multiple ports have the same priority, the booting platform may select the port in any order relative to the ports that have this same priority.

A5.6.4.3 CONSOLEPORTPRIORITY

ConsolePortPriority identifies which ports are more likely to find the Console. This component is useful when the booting platform attaches to multiple subnets. When the booting platform processes a ConsoleLocatorRecord then the booting platform should access the Console through the port with the highest priority first. If that fails, the booting platform proceeds with lower priority ports in priority order. When multiple ports have the same priority, the booting platform may select the port in any order relative to the ports that have this same priority.

A5.6.4.4 IOCPORTPRIORITY

IocPortPriority identifies which ports are more likely to find the IOC from which to load the OS boot loader. This component is useful when the booting platform attaches to multiple subnets. When the booting platform processes a OsLocatorRecord then the booting platform should access the OS boot loader through the port with the highest priority first. If that fails, the booting platform proceeds with lower priority ports in priority order. When multiple ports have the same priority, the booting platform

may select the port in any order relative to the ports that have this same priority.

A5.6.4.5 NETWORKBOOTPORTPRIORITY

NetworkBootPortPriority identifies which ports are more likely to find the IB Network boot server. This component is useful when the booting platform attaches to multiple subnets. When the booting platform processes a OsLocatorRecord then the booting platform should access the OsLocatorRecord through the port with the highest priority first. If that fails, the booting platform proceeds with lower priority ports in priority order. When multiple ports have the same priority, the booting platform may select the port in any order relative to the ports that have this same priority.

A5.6.4.6 TIME-OUTS

Three components provide the booting platform with time-out values used in determining how much time a platform should wait for subnet resources (InitTimeout), the BIS (BisTimeout) and end nodes (EndNodeTimeout) to become operational.

A5.6.4.6.1 INITTIMEOUT AND BISTIMEOUT

Determining the amount of time after power-on-reset to contact an end-node providing a boot device, console, or BIS service is the role of the InitTimeout and BisTimeout values. Each time-out specifies the minimum amount of time in 100 mSec increments (approximately 109 minutes max.). The booting platform waits at least this period of time before making the determination that the node is not available. The timer begins at power on reset of the booting platform and unconditionally counts. The BootManager should set this value to the maximum amount of time it takes for IOUs or Services to become operational and accept requests from booting platforms.

When the booting platform is connecting to a device or console service, the platform continues trying to connect to the IOC/Service for at least the period of time listed in InitTimeout before declaring that the IOC or service is not operational through this port.

If the booting platform queries the BIS, then the platform continues to query the BIS for at least the period of time listed in BisTimeout before declaring that the BIS is not operational through this port.

The InitTimeout value allows sufficient time for the all IOCs or services to become available to the booting platform.

For BIS, the time-out value provides sufficient time for the BIS Service to boot itself, register with the SA, and become available on the subnet.

Generally, a booting platform attempts to communicate with an endnode for booting. If the platform cannot communicate with the endnode, time-out values are used, and only then is the Free Running Timer compared against the time-out value. This allows booting platforms to make progress as soon as the resource becomes available on the network. Once the time-out value has been reached, the BtA tries to use each Locator Record in priority order with zero tolerance for “no path” reported by the SA. That is, once the SA reports “no path” the BtA moves on to the next port and then the next Locator record.

oA5-45: The boot platform shall contain a Free Running Timer that is initialized at power on reset and counts time unconditionally.

oA5-46: If the booting platform supports Traps, the platform shall issue a BootReport Trap when a failure listed in Table 382 on page 1343 occurs.

oA5-47: If the booting platform supports Notices, it shall post a BootReport notice when a failure listed in Table 382 on page 1343 occurs.

There may be a need to update time-out values periodically as fabric response times slow down due to scaling factors, such as increasing the number of platforms booting from the same IOU. As the number of booting platforms sharing resources increases, it is expected that booting will take longer. As it takes longer to boot it may be undesirable that a booting platform time-out value is reached due to subnet scaling issues. There are many ways of calculating the correct timer values and preventing the timers from expiring. One might be the BootManager periodically adjusting the value of the timers as a simple function of the number of platforms booting from the same IOU.

A5.6.4.6.2 ENDNODETIMEOUT

Some nodes needed to boot a platform may not be able to respond to MADs immediately upon receiving a MAD. This may be due to the CA being in a powered down state or any other state that might not allow the target node to respond to a MAD.

When the booting platform is connecting to a device or service, then the platform continues trying to connect IOC/Service for at least the period of time listed in EndNodeTimeout before declaring that the IOC or service is not operational through this port. Unlike BisTimeout and InitTimeout, each target (not just the first) is given at least this much time to respond.

oA5-48: The booting platform shall continue to retry MADs sent to target nodes for a period of time defined in EndNodeTimeout or the node has generated a response.

oA5-49: If Traps are supported the booting platform shall issue Trap 0x0110 during boot if the platform does not receive a Response to a MAD

Request in the period specified in section 13.4.6.2 Timers and Timeouts on page 727 and the platform has given up retrying the MAD.

oA5-50: If Notices are supported the booting platform shall post notice 0x0110 during boot if the platform does not receive a Response to a MAD Request in the period specified in section 13.4.6.2 Timers and Timeouts on page 727 and the platform has given up retrying the MAD.

A5.6.5 PERSISTENT LOCATOR RECORDS

The BootManager can Get() and optionally Set() Local-Persistent LocatorRecords. When a BootManager does a Set() to OsLocatorRecord, ConsoleLocatorRecord, or RomRepositoryLocatorRecord, then the BtA saves the record persistently. The BtA indicates which Locator Records that the booting platform supports and whether the BootManager can modify them in the PlatformBootInfo:Capability component (see A5.6.3 "PlatformBootInfo Attribute" on page 1301). The booting platform uses the Locator Records the next time the platform boots using Local-Persistent Locator Records. The BootManager can determine the current set of persistent Locator Records of a booting platform by issuing Get()s to that platform's BtA Locator Record attributes.

There are 3 Local-Persistent Locator Record types:

- ROM Repository - Pointer to the ROM Repository
- Console - Pointer to the Console
- OS - Pointer to the OS boot loader

Each booting platform saves Persistent Locator Records in slots. The number of slots available per record type is defined by the PlatformBootInfo Capability and Count components. Slot numbers can range from 0 to 255 with slot 0 being the highest priority and 255 the lowest. The AttributeModifier component in the MAD header serves as an index to the slot. For example, BootMgtSet(RomRepositoryLocatorRecord) with AttributeModifier=0 writes slot 0 of the RomRepositoryLocatorRecord. AttributeModifier=1 writes slot 1 of the RomRepositoryLocatorRecord, and so on. AttributeModifier is used for both Set() and Get() to the RomRepositoryLocatorRecord, ConsoleLocatorRecord, and OsLocatorRecord attributes.

.

Table 373 RomRepositoryLocatorRecord Attribute

Component	Access	Offset (bits)	Length	Description
Reserved	RO	0	16-Bytes	Reserved

Table 373 RomRepositoryLocatorRecord Attribute (Continued)

Component	Access	Offset (bits)	Length	Description
PortGID	R/W	128	16-Bytes	Port GID for the IOU
Reserved	RO	256	160-Bytes	Reserved

.

Table 374 ConsoleLocatorRecord Attribute

Component	Access	Offset (bits)	Length	Description
Reserved	RO	0	8-bits	Reserved
Device-Service	R/W	8	1-bit	Indicates if this record describes a Console Device or Server: 0b = this record describes an Console device 1b = this record describes a Console server
DeviceDriverLocation	R/W	9	2-bits	Provides information on the search order to locate a Device Driver supporting this Console: 00b = Search only the ROM Repository on the same IOU as the IOC 01b = First search the ROM Repository on the same IOU as the IOC then use the RomRepositoryLocatorRecords 10b = First search the ROM Repository pointed to by the RomRepositoryLocatorRecords then use the ROM Repository on the same IOU as the IOC 11b = Search only the ROM Repository pointed to by the RomRepositoryLocatorRecords
Reserved	RO	11	5-bits	Reserved
Protocol	R/W	16	8-bits	Identifies the console protocols supported by the console device or service: • 0x00 = unknown otherwise bit specific where: • bit 0 - proprietary protocol • bit 1 - IBTA Console protocol (refer to Annex A2: Console Service Protocol on page 1140) • bits 2-6 are reserved and set to zero • bit 7 -Use any ProtocolName specified by a ProtocolName element in the AdditionalInfo component.
Reserved	RO	24	5-Bytes	Reserved

Table 374 ConsoleLocatorRecord Attribute (Continued)

Component	Access	Offset (bits)	Length	Description
IocGUID-SID	R/W	64	8-Bytes	GUID of the I/O controller or ServiceID of the service depending on setting of the Device-Service component • If Device-Service = 0b (console IOC), then this is the IocGUID. • If Device-Service = 1b (server process), then this is the ServiceID.
PortGID	R/W	128	16-Bytes	Port GID for the IOU or service
AdditionalInfo	R/W	256	160 Bytes	Console Protocol specific data in Type-Length-Value (TLV) format see A5.11 "AdditionalInfo" on page 1362. This component is passed to the Console Driver.

.

Table 375 OsLocatorRecord Attribute

Component	Access	Offset (bits)	Length	Description
BootMethod	R/W	0	8-bits	Indicates the Boot Method: • 0x01 = Storage (Locator Record specifies IOC to use) • 0x02 = Network (Locator Record specifies IOC) • 0x03 = Proprietary (Locator Record specifies IOC to use) • 0x04 = IB Network Boot (Locator record specifies IB network boot protocol and/or IB Boot Server, i.e., it does not specify an IOC) all other values reserved
Reserved	R/W	8	1-bit	reserved
DeviceDriverLocation	R/W	9	2-bits	Provides information on the search order to locate a Device Driver supporting this Locator Record: 00b = Search only the ROM Repository on the same IOU as the IOC 01b = First search the ROM Repository on the same IOU as the IOC, then use the RomRepositoryLocatorRecords. 10b = First search ROM Repositories pointed to by the RomRepositoryLocatorRecords, then use the ROM Repository on the same IOU as the IOC. 11b = Search only the ROM Repositories pointed to by the RomRepositoryLocatorRecords.

Table 375 OsLocatorRecord Attribute (Continued)

Component	Access	Offset (bits)	Length	Description
RecordFunction	RO	11	2-bits	Indicates the purpose of this Locator Record: • **00b - Install Source** - this record describes a source for the installation program which can install an OS for the booting platform. • **01b - Boot Source** - this record describes a source for the booting platforms OS boot loader • **10b - Install Destination** - this record describes a destination where an installation program can install an OS boot loader. A booting platform should not attempt to use an OsLocatorRecord with RecordFunction=10b as the source for its boot loader. • **11b - Install Destination/Boot Source** - this record describes a destination where an installation program can or has installed an OS boot loader. A booting platform can attempt to boot from this location. An installation program can install an OS boot loader at this location.
Reserved	RO	13	3-bits	Reserved
Protocol	R/W	16	8-bits	Specifies I/O protocols supported by the IOC • 0x00 = unknown otherwise bit specific (refer to Table 376 "Protocol Component Bit Definitions" on page 1331):
Reserved	RO	24	5-bytes	Reserved
IocGUID/SID	R/W	64	8-Bytes	GUID of the I/O controller or SID of the IB Network Boot Service (a value of zero means unknown or not used).
PortGID	R/W	128	16-Bytes	Port GID for the IOU or the IB Network Boot Service (a value of zero means unknown or not used)
AdditionalInfo	R/W	256	160 Bytes	Protocol specific data in Type-Length-Value (TLV) format which identifies the device or path behind the IOC or the IB Network Boot Service. This component is passed to the IO Driver. See A5.11 "AdditionalInfo" on page 1362.

When multiple BootManagers access the same booting platform the BootManagers should act in concert. To ensure BootManagers are aware of changes, when a LocatorRecord is changed then the BootAgent sends the BootManager identified in each port's ClassPortInfo, a ChangeReport Trap informing them of the change.

A5.6.5.1 DEVICE-SERVICE

This component qualifies the IocGUID/SID component. If Device-Service = 0b, then the IocGUID/SID component contains the IocGUID. Otherwise it contains the ServiceID.

A5.6.5.2 IOCGUID-SID

This component defines the GUID of the IOC if the Device-Service component indicates a device. If the Device-Service component indicates a service this component contains a ServiceID. For IB network boot, the SID identifies the boot server (or the first server that the booting platform contacts). A value of zero is valid if the network boot protocol is capable of finding its own boot server. The AdditionalInfo component provides protocol specific information to further help the booting platform locate the protocol specific objects.

A5.6.5.3 PORTGID

PortGID of the CA where the IOU or Service resides.

A5.6.5.4 PROTOCOL

Locator records contain a Protocol component that the booting platform uses to determine which protocols the booting platform can use with the specified device. This field is bit specific (i.e., each bit represents a protocol). When a device supports more than one protocol, the BootManager can indicate a preference by setting multiple records, each with a single Protocol value. If the BootManager sets more than one bit, it means that the booting platform may use any of the indicated protocols. A value of zero means that the BootManager does no know which protocols the device supports. In this case the booting platform determines which protocol to use.

Protocols are dependant on the BootMethod component as specified in are Table 376.

Table 376 Protocol Component Bit Definitions

Bit	Boot Method			
	01 Storage	**02 LAN Network Boot**	**03 Proprietary**	**04 IB Network Boot**
0	proprietary	proprietary	proprietary	proprietary
1	SRP	Reserved	Reserved	IPoIB
2-6	Reserved			
7	Use any ProtocolName specified by a ProtocolName element in the AdditionalInfo component.			

A5.6.5.5 RECORDFUNCTION

Local-Persistent and BIS OsLocatorRecords are provided to the booting platform in priority order. When trying to load the OS, a booting platform processes OsLocatorRecords starting with the first record and continuing until it finds a boot loader. The boot loader can be the normal OS boot

loader that loads the OS onto the platform or a boot loader that loads an OS-Install program onto the platform. The RecordFunction component identifies the intent of the OsLocatorRecord as (a) a pointer to the source of the platforms OS boot loader, (b) a pointer to the source of the platforms OS-Install boot loader, and/or (c) a pointer to the destination device used by the OS installation program as the target for a newly installed OS boot loader.

By including an OsLocatorRecord with RecordFunction = 00b, the BootManager or BIS enables the installation process. Setting RecordFunction = 01b identifies that the OS loader is to be used as a source but ignored by the OS-Install program. Setting RecordFunction=10b identifies that the OsLocatorRecord only identifies a destination where the installation program can install an OS boot loader and is ignored by a platform as a boot loader source. Setting RecordFunction=11b identifies that the OsLocatorRecord is a source for the platforms OS boot loader and a destination where the installation program can install an OS boot loader. Thus this record can be used by the platform when it is trying to boot and by the Installation program.

The order in which the BootManager or BIS provides OsLocatorRecords effects the booting process. For example, providing an OsLocatorRecord that points to an OS-Install Program (RecordFunction=00b) first followed by the second OsLocatorRecord that points to the destination device (usually RecordFunction=11b) can be used to install or re-install an OS. Supplying multiple source and destination OsLocatorRecords provides redundancy.

If an OS-Install is not desired then the BootManager or BIS provides only OsLocatorRecords that describe OS boot loader sources (RecordFunction = 01b) and does not provide OS-Install sources, RecordFunction=00b.

If the policy is to attempt to boot an OS and upon detection of a missing or corrupt boot loader reinstall the OS onto a new device then the BootManager or BIS could provide two OsLocatorRecords, the first OsLocatorRecord (RecordFunction =11b, both destination and source) and the second OsLocatorRecord describing the source of a boot loader and an OS-Install Program (RecordFunction=00b). If the booting platform detects a failure in the first source, such as failing to find a valid boot sector on the disk or detects that the OS boot loader was not installed properly, then it proceeds to the next OsLocatorRecord and thus uses the OsLocatorRecord RecordFunction=00b as source to install the OS. The installation program then uses OsLocatorRecord (RecordFunction =11b) as the target for the new OS. If the installation was successful, then the next time the platform reboots, it finds a valid OS boot loader at the location specified by the first OsLocatorRecord and the booting platform boots the OS

from the first OsLocatorRecord and would not use the OsLocatorRecord that re-installs the OS.

OsLocatorRecords can be provided so that the platform normally attempts to load the OS, but if it encounters a device failure while attempting to load the OS, the platform will install the OS onto another device To do this the BootManager or BIS would provide three OsLocatorRecords (A,B and C). The first OsLocatorRecord, RecordFunction =01b, record A points to the source of the active boot loader. The second OsLocatorRecord, record B describes the source of an OS Install Program (RecordFunction=00b). The third OsLocatorRecord, RecordFunction =11b, record C points to a backup boot loader destination. If the booting platform detects a device failure in the first source (A) in which the platform cannot access to the boot loader, then it bypasses this OsLocatorRecord and proceeds to the second source OsLocatorRecord (B) as source to install the OS. The installation program then uses the third destination OsLocatorRecord, RecordFunction =11b, record (C) as the target of the newly installed OS. Care should be taken that this platform does not repeatedly install the OS. In this case, during subsequent reboots, the booting platform may have the ability to determine that the new OS has been installed correctly and by-pass repetitive re-installs of the OS. Alternatively, the booting platform could wait for the BootManager to update OsLocatorRecords by either removing the OsLocatorRecord pointing to the OS-Install or placing the newly installed OS locator at a higher priority than the OS-Install OsLocatorRecord.

Another possibility is to provide source OsLocatorRecords before any OS-Install locator records such that the OS-Install only occurs after the boot environment attempts to boot from all source devices. To do this the BootManager or BIS could provide three OsLocatorRecords. The first OsLocatorRecord, RecordFunction =01b, points to the source of the active boot loader. The second OsLocatorRecord, RecordFunction =11b, points to a device that is both the destination of an OS-Install and a boot source which does not currently contain a boot loader. The third OsLocatorRecord, points to the OS Install Program (RecordFunction=00b). With this combination, the first time device pointed to by the first record fails to produce an OS loader then the booting platform skips loading from the device pointed to by the first locator record. The booting platform also skips the second record because it does not yet have an OS Loader. Thus, the booting platform uses the third record as the OS-Install source and uses the second record as a pointer to the destination of the OS-Install. After the OS installation completes and the platform re-boots, the first record again fails to produce a boot loader and the platform boots from the second record, which now contains a valid OS loader.

A5.6.6 NODE REBOOT

The system administrator using the BootManager may reboot a platform by issuing a Set(NodeReboot) MAD. Normally, the system administrator would use a graceful reboot to shutdown the OS in an orderly fashion. There are some conditions that may occur that prevent a graceful reboot from completing. Under this circumstance the system administrator may need to signal an immediate reboot to the platform. It is the boot platform that makes the final determination of the actions that it takes when it receives a NodeReboot. The RebootStatus in the BootMgtGetResp() indicates the platforms reaction to the Reboot MAD.

Varying platforms offer different types of reboots. Some might clear caches and memory and some might not. Some might run a number of diagnostics and some might not. What is of concern is the state of the BtA when the reboot occurs. It is expected that if the platform supports IB reboot, that it not clear the information required to reboot. Of particular interest are the R/W components in PlatformBootInfo and the Locator Records themselves. These must be maintained by the booting platform in order for the reboot to be successful.

A5.6.6.1 NODEREBOOT ATTRIBUTE

Table 377 NodeReboot Attribute

Component	Access	Offset (bits)	Length	Description
Type	R/W	0	1bit	Reboot Type • 0b - Graceful Reboot - Shutdown and Reset. Close files etc., then reboot. • 1b - Immediate Reboot - Immediate Reboot. This component is reserved as zero in Resp() to a BootMgtGet(NodeReboot);
Reserved	RO	1	5-bits	Reserved
LocatorRecordType	R/W	6	2-bits	Select the Locator Record Type • 00b - Other - Non-IBA boot (e.g., from PCI) • 01b - Selects Persistent or BIS record as set in PlatformBootInfo:OsBootLocator. • 10b - reserved • 11b - reserved This component is set to zero in the BootMgtGetResp() to a BootMgtGet(NodeReboot);
RebootStatus	RO	8	8-bits	Reboot Status • 0x00 - Not attempting to Reboot • 0x01 - Reboot in progress • 0x02 - Reboot failed

Table 377 NodeReboot Attribute (Continued)

Component	Access	Offset (bits)	Length	Description
Reserved	RO	1	16-bits	Reserved
Timestamp	RO	32	32-bits	Timestamp - The BtA reports the time that the last reboot started. The timestamp is the number of seconds since 1/1/70 00:00am.
Reserved	RO	64	184 Bytes	Reserved

oA5-51: If Capability(14:15) is non-zero, then the BtA shall report the time that the last reboot started (or the time that the BtA was loaded if no reboot) in the NodeReboot:Timestamp component in a format that represents the number of seconds from 1/1/70 00:00am in NodeReboot:Timestamp.

oA5-52: If Capability(15) is 1b then the BtA shall request the platform (OS, BIOS, etc.) to gracefully reboot the platform when a BootMgtSet(NodeReboot:Type) = 0b is received from any port.

oA5-53: If Capability(14) is 1b then the BtA shall request the platform (OS, BIOS, etc.) to immediately reboot the platform when a BootMgtSet(NodeReboot:Type) = 1b is received from any port.

oA5-54: If Capability(14:15) is non-zero, then the booting platform shall use the value in NodeReboot:LocatorRecordType to select the source of the OS record locator.

oA5-55: If Capability(14:15) is non-zero then the BtA shall return BootMgtGetResp(NodeReboot) according to Table 377 "NodeReboot Attribute" on page 1334.

A5.6.6.2 REBOOT TIME LINE

Figure 275 on page 1336 illustrates that two BtAs (non-OS and OS BtAs) can exist on a booting platform. After a booting platform is powered on, it takes some period of time for a BtA to become active (Blue bar). This non-OS BtA normally exists on the booting platform before the OS is loaded and is part of the Platform Vendors boot environment. It is the responsibility of the boot environment and the Non-OS BtA to locate boot devices containing the Console and OS boot loaders. Many times, when the platform vendors boot environment gives up control to the OS loader there is a period of time when a BtA is not available on the port. Then as the OS becomes functional, an OS BtA may appear in place of the non-OS BtA. The non-OS and OS BtAs may have differing capabilities. For example,

the non-OS BtA might have access to non-volatile storage and the OS BtA might not.

The BootMgt GetResp(NodeReboot:Status) reflects the progress of the platform with respect to the reboot. Although a reboot is shown occurring while the OS BtA is running, a non-OS BtA may also be requested to reboot.

Traps or Notices, if supported, are very useful in assisting the BootManager in determining the state of the booting platform. For more information on Traps see A5.6.7 "Traps and Notice Queues" on page 1336.

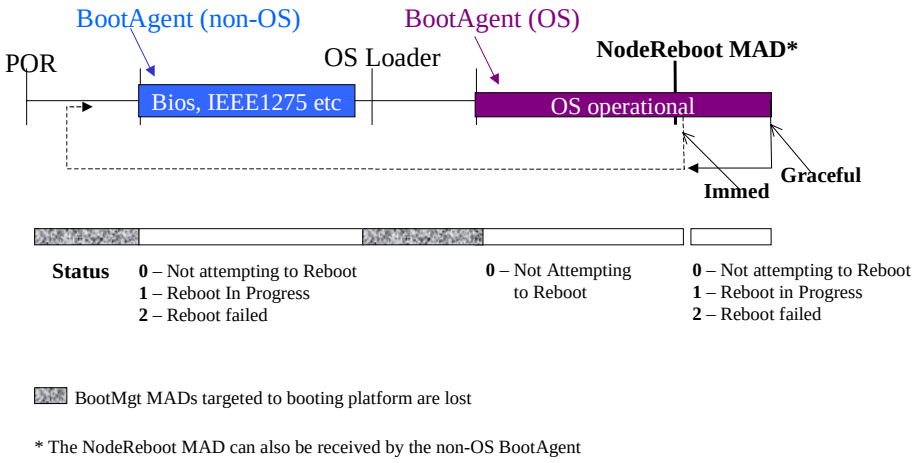


Figure 275 Reboot Time Line

A5.6.7 TRAPS AND NOTICE QUEUES

Traps and Notices Queues are optional and allow a BtA to inform a BootManager of failures while the platform is booting or at anytime when a failure is detected which might affect a future booting of the platform. Some notable traps or notices are a failed attempt to locate an IOU and the loss of Local-Persistent Locator Records within the booting platform. Once a failure is detected, it should be reported to the BootManager via the trap or notice mechanism.

Traps and Notice Queues are described in detail in 13.4.9 Traps on page 741 and 13.4.10 Notice Queue on page 743. This annex uses the Common Framework described in Chapter 13 and this section serves only to describe behavior that is unique to the Boot Management class.

The BootManager shall [Compliance assured in the base document] support Traps and Notices. Traps are asynchronous notifications for the purpose of alerting an entity about exception conditions or other events of interest related to booting. The BootManager can also use a BootMgtReport(Heartbeat) to inform subscribed clients that the Boot Manager is still operational.

Other 3rd party nodes may want to register for Event Forwarding to be notified when a booting Platform reports a boot failure or boot event.

The Trap and Notice ladder diagrams in Figure 276 on page 1337 and Figure 277 on page 1338 respectively describe the flow of reporting failures to/from the BtA and to/from the BootManager.

Boot Management Trap Handling

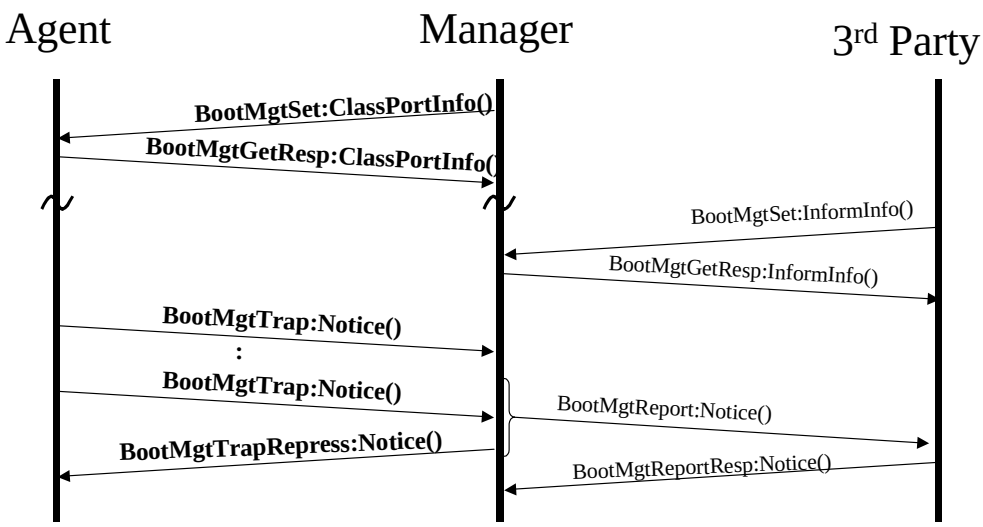


Figure 276 Boot Management Trap Flow

Boot Management Notice Handling

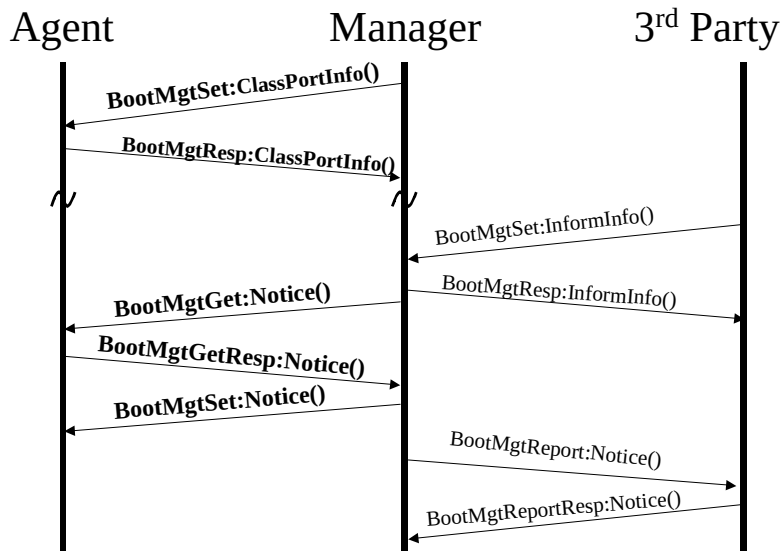


Figure 277 Boot Management Notice Flow

For details on Trap and Notice usage refer to 13.4.9 Traps on page 741 and 13.4.10 Notice Queue on page 743.

In each Data Details definition there is a LostTrap bit. This bit is used to signal when the boot agent had to stop sending a trap before receiving a TrapRepress(). The premise is that the Boot Agent has a trap queue of limited depth (one or more outstanding traps). When a new trap is generated, it is placed at the tail of the trap queue and the trap at the head of the queue is repeated until the Boot Agent receives a matching TrapRepress(). The matching trap is then discarded and then next trap in the queue is sent immediately and repeated until a matching TrapRepress is received.

If the trap queue is full and a new trap is generated, the Boot Agent discards the trap at the head of the trap queue, adds the new trap to the tail of the trap queue, sets the LostTrap bit in trap at the head of the queue and immediately starts sending that trap.

There could be other reasons why a trap is discarded. It is the boot agent's responsibility to set the LostTrap bit in the first trap sent after the discarded trap would have been sent. If the discarded trap is the only trap in the queue, then the agent should fabricate a StatusReport trap with StatusType = 0x00 (Trap or Notice Queue flushed) and the LostTrap bit set to indicate the lost trap.

This bit is also used to signal when the boot agent had to discard a trap in the Notice Queue before receiving a Set(Notice). With the Notice Queue, when a new trap is generated, it is placed at the tail of the queue and the Trap at the head of the queue is read until the Boot Agent receives a matching Set(Notice). When the BtA receives a matching Set(Notice), it discards the matching trap and returns the next trap in the queue in the GetResp(Notice). If the Notice Queue is full when a new trap is generated, the Boot Agent discards the trap at the head of the Notice Queue, adds the new trap to the tail of the queue, and sets the LostTrap bit in the trap at the head of the queue.

There might be other events that could cause the boot agent to prematurely discard one or more notices. When there is at least one notice to be reported and a condition occurs in which the BtA discards all of the notices, then the BtA generates a StatusReport notice with a StatusType of x00 and sets the LostTrap. This will inform the Boot Manager and subscribed clients that one or more notices have been lost.

A5.6.7.1 NOTICE ATTRIBUTE

The Notice attribute describes an exception or other CA, switch, or router event. It is used by both the trap mechanism described in 13.4.9 Traps and the Notice mechanism described in 13.4.10 Notice Queue.

Table 378 Notice Attribute

Component	Access	Offset (bits)	Length (bits)	Description
BaseNoticeComponents	RO	0	80	This component contains the IsGeneric, Type, NodeType, TrapNumber, IssuerLID, NoticeToggle and NoticeCount defined in section 13.4.8.2 Notice on page 737
DataDetails	RO	80	432 (54 Bytes)	Each Trap defines its own Data Details. See Table 380 "Notice Details for Trap 0x0000 - KeyViolation" on page 1341, Table 381 "Notice Details for Trap 0x0100 - ChangeReport" on page 1342, Table 382 "Notice Details for Trap 0x0110 - StatusReport" on page 1343, and Table 383 "Notice Details for Trap 0x0007 - Heartbeat" on page 1345

CA5-39.2.1: Unless otherwise specified, Notice attribute components for Notices specified in Table 379 shall be set as follows:

- IsGeneric = 1 (Generic)
- Type - as per Table 379
- ProducerType = 1 (Channel Adapter)
- TrapNumber - as per Table 379

- IssuerLID - as per Section 13.4.8.2, “Notice,” on page 737
- NoticeToggle - as per Section 13.4.8.2, “Notice,” on page 737
- Notice Count - as per Section 13.4.8.2, “Notice,” on page 737
- DataDetails - as per Table 379
- IssuerGID=0 as per Section 13.4.8.2, “Notice,” on page 737

oA5-56: If the BtA supports Traps or the Notice Queue, then the BtA shall use the format in Table 378 "Notice Attribute" on page 1339, Table 380 "Notice Details for Trap 0x0000 - KeyViolation" on page 1341 and Table 381 "Notice Details for Trap 0x0100 - ChangeReport" on page 1342.

This section specifies specific traps for the Boot Management Class. DataDetails of the Notice attribute leaves 54 bytes to trap specific data (See DataDetails in Table 378 on page 1339).

Table 379 Boot Management Traps

Trap Name	Type	TrapNumber	DataDetails
KeyViolation	Security	0x0000	See Table 380 "Notice Details for Trap 0x0000 - KeyViolation" on page 1341
ChangeReport	Urgent	0x0100	See Table 381 "Notice Details for Trap 0x0100 - ChangeReport" on page 1342
StatusReport	Urgent/ Informational	0x0110	See Table 382 "Notice Details for Trap 0x0110 - StatusReport" on page 1343
Heartbeat	Informational	0x0007	Indicates that the Boot Manager is still active. See Table 383 "Notice Details for Trap 0x0007 - Heartbeat" on page 1345
All other Boot Management Traps are reserved.			

oA5-57: If the BtA supports Traps or the Notice Queue, then the BtA shall only use the Type and Number listed in Table 379 on page 1340.

The BtA sends traps to the BootManager listed in ClassPortInfo.

oA5-58: The BtA shall zero the BtM_Key component when generating a BootMgtTrap().

A5.6.7.1.1 KEYVIOLATION NOTICE

Table 380 Notice Details for Trap 0x0000 - KeyViolation

Component	Offset (bits)	Length (in bits) 54B/432b	Description
BtAType	0	8b	**0x00** - BtA is Firmware controlled, non-Extended **0x01** - BtA is Firmware controlled, Extended **0x10** - BtA is OS controlled All other encodes are reserved
Method	8	8b	Method used in MAD that caused the violation
AttributeID	16	16b	AttributeID used in MAD that caused the violation
ViolatedPortGUID	32	64b	Port GUID of the port receiving the violation
OffendingPortGid	96	128b	Requestor port GID from the MAD that caused the violation
OffendingLIDADDR	224	16b	SLID used in MAD that caused the violation
Reserved	240	16	Reserved
QP	256	24b	Source Queue Pair from the MAD that caused the violation
Reserved	280	8b	Reserved
BtM_Key	288	64b	BtM_Key from the MAD that caused the violation
Timestamp	352	48b	Timestamp when violation was detected. The format of the timestamp is the number of milliseconds since 1/1/70 00:00am Zero indicates Timestamp not supported
Reserved	384	47b	Reserved
LostTrap	431	1b	This bit is set when the BtA had to discard a Trap before receiving a TrapRepress() for it. The bit is reset when the IOU receives a valid TrapRepress() with this bit set.

oA5-59: If the BtA supports Traps as indicated in ClassPortInfo, then the BtA shall set the KeyViolation notice as specified in Table 380 on page 1341 and issue the Trap to every port when a BtM_Key mismatch is detected.

oA5-60: If the BtA supports Notices as indicated in ClassPortInfo, then the BtA shall post the KeyViolation notice as specified in Table 380 on page 1341 to the notice queue when a BtM_Key mismatch is detected.

oA5-61: If the BtA generates a trap, then a trap shall be issued to all ports that have a non-zero ClassPortInfo:TrapLID.

oA5-62: "If the BootAgent supports notices, it shall maintain a a single notice queue accessible from every port on the platform.

A5.6.7.1.2 CHANGEREPORT NOTICE

Table 381 Notice Details for Trap 0x0100 - ChangeReport

Component	Offset (bits)	Length (in bits) 54B/432b	Description
BtAType	0	8b	**0x00** - BtA is Firmware controlled, non-Extended **0x01** - BtA is Firmware controlled, Extended (See A5.1.8.3 "Boot Environment Extension" on page 1281) **0x10** - BtA is OS controlled All other encodes are reserved
Method	8	8b	Method used in MAD that caused this event
BootMgrLID	16	16b	BootManager LRH:SLID that caused this event
AttributeID	32	16b	AttributeID used in MAD that caused this event (see Table 366 on page 1293)
Reserved	48	16	Reserved
PlatformPortGUID	64	64b	GUID of the port on the booting platform receiving the BootMgtSet() that caused this event
BootMgrPortGID	128	128b	GRH:SGID of the port GID that caused this event. This value is zero if the MAD does not contain a GRH.
AttributeModifier	256	32b	AttributeModifier used in MAD that caused this event
Reserved	288	8b	Reserved
BootMgrSrcQP	288	24b	Source Queue Pair (DETH:SrcQP) from the MAD that caused this event
Timestamp	320	48b	Timestamp when change was detected. The format of the timestamp is the number of milliseconds since 1/1/70 00:00am Zero indicates Timestamp not supported
Reserved	368	63	Reserved
LostTrap	431	1b	This bit is set when the BtA had to discard a Trap before receiving a TrapRepress() for it. The bit is reset when the IOU receives a valid TrapRepress() with this bit set.

oA5-63: If the BtA supports Traps, then any change made by a proprietary mechanism which affects the outcome of a platform boot for conditions specified in Table 381 on page 1342 shall be accompanied by a ChangeReport Trap 0x0100 issued on every port.

oA5-64: If the BtA supports the Notice Queue, then any change made by a proprietary mechanism which affects the outcome of a platform boot for conditions specified in Table 381 on page 1342 shall be accompanied by a 0x0100 notice posted to the Notice Queue.

A5.6.7.1.3 STATUSREPORT NOTICE

Table 382 Notice Details for Trap 0x0110 - StatusReport

Component	Offset (bits	Length (in bits) 54B/432b	Description
BtAType	0	8b	**0x00** - BtA is Firmware controlled, non-Extended **0x01** - BtA is Firmware controlled, Extended (See A5.1.8.3 "Boot Environment Extension" on page 1281) **0x10** - BtA is OS controlled All other encodes are reserved)
LocatorRecordType	8	8b	**0x00** - none **0x02** - Local-Persistent OsLocatorRecord **0x03** - BIS OsLocatorRecord **0x04** - Local-Persistent ConsoleLocatorRecord **0x05** - BIS ConsoleLocatorRecord **0x06** - Local-Persistent RomRepositoryLocatorRecord **0x07** - BIS RomRepositoryLocatorRecord All other encodes are reserved
LocatorRecordNumber	16	8b	If LocatorRecordType = 0x02-0x07 then the LocatorRecordNumber identifies the locator record of interest. If Local-Persistent Locator Record - AttributeModifier used in the Set(). If BIS Locator Records - this value represents the priority order by which the BIS sent locator records back to the booting platform in response to a BIS query. The first record is 0x00, the second is 0x01, and so on. If LocatorRecordType is other than 0x02-0x07, this value should be set to 0x00.

Table 382 Notice Details for Trap 0x0110 - StatusReport (Continued)

Component	Offset (bits	Length (in bits) 54B/432b	Description
StatusType	24	8b	Reason for StatusReport **0x00** - Trap Queue or Notice Queue flushed **0x01** - Platform Boot Failed, reboot- platform will initiate reboot **0x02** - Platform Boot Failed, waiting for BootManager to initiate reboot via Set(Reboot) **0x03** - Platform Boot Failed, Boot canceled manually **0x04** - Platform Boot Failed, reboot - platform will initiate non-IB boot process **0x05** - reserved **0x06** - Non-volatile platform storage failure **0x07** - Locator Record Failure (see LocatorRecordType) **0x08** - Extended ROM Repository Failure **0x09** - Device Driver ROM Repository Failure **0x0A** - IOU Failure **0x0B** - Network Boot server failure **0x10** - No path to BIS - SA did not return PathRecord **0x11** - BIS Timeout - BIS did not return Locator Records in allowed time (see BisTimeout in Table 372 "PortBootInfo Attribute" on page 1321 **0x12** - BisQueryResp() received without matching TransactionID - see BisPortGID for failing BIS **0x13** - ProtocolName not found in AdditionalInfo. This is set when the OsLocatorRecord:Protocol(7)=1b and OsLocatorRecord:Addition-alInfo does not contain a protocol. **0x14** - ProtocolName not found in AdditionalInfo. This is set when the ConsoleLocatorRecord:Protocol(7)=1b and OsLocator-Record:AdditionalInfo does not contain a protocol. **0x20** - BtA started **0x21** - BtA terminated, OS Loading - Persistent Boot **0x22** - BtA terminated, OS Loading - BIS Boot **0x23** - reserved **0x24** - BtA terminated, OS Loading - Non-IB Boot **0x25** - BtA terminated, Reboot
Reserved	32	24b	Reserved
AdditionalStatus	56	8b	Additional Status **0x00** - No additional status **0x01** - SA did not return a Path to the targeted port **0x02** - MAD Time-out **0x03** - Connection Time-out **0x04** - ROM Repository image not found **0x05** - ROM Repository image failed (i.e. platform determined the image is not usable) **0x06** - Connection to target failed - Service not found **0x07** - Connection to target failed - Connection failure

Table 382 Notice Details for Trap 0x0110 - StatusReport (Continued)

Component	Offset (bits	Length (in bits) 54B/432b	Description
Timestamp	68	48b	Timestamp when error was detected. The format of the timestamp is the number of milliseconds since 1/1/70 00:00am Zero indicates Timestamp not supported
BisPortGID	112	128b	PortGID of the BIS() if BIS failure else set to zero
Reserved	240	191	Reserved
LostTrap	431	1b	This bit is set when the BtA had to discard a Trap before receiving a TrapRepress() for it. The bit is reset when the IOU receives a valid TrapRepress() with this bit set.

oA5-65: If the BtA supports Traps as indicated in ClassPortInfo, then the BtA shall issue a StatusReport Trap 0x0110 to every port on the booting platform for conditions described in StatusType or AdditionalStatus in Table 382 on page 1343.

oA5-66: If the BtA supports Notices as indicated in ClassPortInfo, then the BtA shall post a StatusReport notice 0x0110 in the Notice Queue for conditions described in StatusType or AdditionalStatus in Table 381 on page 1342.

A5.6.7.1.4 HEARTBEAT NOTICE

This notice is not generated by the Boot Mgt agent, it is generated by a Boot Manager in a BootMgtReport() to inform subscribed clients that the Boot Manager is still operational.

Table 383 Notice Details for Trap 0x0007 - Heartbeat

Field	Offset (bits)	Length (bits)	Description
TTNH	0	12	Time till next heartbeat - specifies the number of minutes before the Boot Manager will send another Heartbeat notice. If more that this time elapses, it is an indication that the Boot Manager has terminated the subscription.
reserved	12	4	
Fail-over	16	1	When this bit is set to one it indicates that a standby manager has taken-over. The client platform should query the SA to locate and verify the new Boot Manager.
reserved	17		

Table 383 Notice Details for Trap 0x0007 - Heartbeat (Continued)

Field	Offset (bits)	Length (bits)	Description
reserved	431	1	LostTrap: This bit is not used and set to zero since this notice is not sent in a Trap() (only in a Report).

See A5.6.7.3.3 "Heartbeat" on page 1350 for requirements on generating the Heartbeat notice.

A5.6.7.2 TRAPREPRESS

Common Trap datagrams are described in 13.4.9 Traps on page 741.

oA5-67: Upon receipt of a valid TrapRepress() MAD and independent of the MADHeader:BtM_Key, the BtA shall cease sending the trap which matches the trap identified by the TrapRepress() MAD. A trap being repeatedly sent matches a trap identified in a TrapRepress() MAD when both MADHeader:TransactionID in the trap MAD matches MADHeader:TransactionID in the TrapRepress MAD and the Notice attribute in the trap MAD matches the Notice attribute in the TrapRepress().

A5.6.7.3 TRAP SUBSCRIPTION / REPORTING

A node may subscribe for BootMgt Trap forwarding by issuing a BootMgtSet(InformInfo) to the Boot Manager. The LID Range and GID in the InformInfo attribute indicates for which booting node the client is interested in receiving Report()s. Note that when subscribing and unsubscribing for the Heartbeat, the GID and LID Range components in the InformInfo is irrelevant and thus the client should set GID, LIDRangeBegin, LIDRangeEnd to zero, 0xFFFF, & zero respectively and the Boot Manager shall ignore those components.

When the Boot Manager receives a BootMgtTrap(), it forwards the trap to subscribed parties via the BootMgtReport(Notice) with the exception of traps listed in Table 384: Privileged Traps. Traps in Table 384 are considered private and must not be forwarded except as indicated in the table.

Table 384 Privileged Traps

Trap	Trap
Key Violation	This is a security trap that contains information which might be considered sensitive (such as invalid keys). The Boot Manager may restrict which nodes receive the BootMgtReport() and/or may zero the key component in the Notice attribute. The Boot Manager policy can be on a trap by trap basis. If a node attempts to subscribe to one of these traps individually and the Boot Manager policy is not to forward the trap to that node, then the Boot Manager rejects the BootMgtSet(InformInfo) with MAD Status [8:15] = Policy Reject (see page 1289). However, the Boot Manager accepts a subscription for 'TrapNumber=0xFFFF' (all traps) regardless of whether it forwards the Key Violation trap.

oA5-67.2.1: When a Boot Manager receives a BootMgtTrap() not listed in Table 384 "Privileged Traps" on page 1347, it shall generate a BootMgtReport() to all parties that have subscribed with the Boot Manager for that notice if a GID of the IOU matches the InformInfo:GID or the IOU has a LID that falls in the range of the InformInfo: LIDRangeBegin – LIDRangeEnd.

oA5-67.2.2: When a Boot Manager receives a BootMgtTrap() listed in Table 384 "Privileged Traps" on page 1347, it shall only generate a BootMgtReport() as described in the table.

Because BootMgt Agents are not required to support Traps, the Boot Manager is required to poll notice queues of Boot Agents that do not support notices, and generate BootMgtReport()s for Notices it reads from those notice queue.

oA5-67.2.3: Except for traps listed in Table 384 "Privileged Traps" on page 1347, the Boot Manager shall generate BootMgtReport()s for Notices it reads from a notice queue of a Boot Agent that does not support Traps (as indicated in the Boot agent's ClassPortInfo:Capability bits).

The manager reports traps in the order that it receives them from the BtA. When the manager receives multiple traps from the same BtA, it delivers them one at a time in the order that the traps were received and waits for a ReportResp() from the client before delivering the next Report().

oA5-67.2.4: A Boot Manager shall report Traps from a BtA in the order it receives the traps. However, the Boot Manager is not required to maintain ordering of traps from different BtAs.

oA5-67.2.5: A Boot Manager shall only have one BootMgtReport() outstanding per subscriber per booting node. That is, the Boot Manager shall

wait for a BootMgtReportResp() before sending a subsequent BootMgtReport() for another trap from the same BtA.

A5.6.7.3.1 SUBSCRIPTION INTEGRITY

Clients depend on receiving Report()s whenever a BtA generates a trap and thus depend on the Boot Manager reporting traps to subscribers. However, there are a number of events that can cause a subscription to be destroyed where the client might not be aware that its subscriptions were destroyed.

- Boot Manager reset - In the event that the Boot Manager is reset, subscriptions can be lost. It is recommended that the Boot Manager retain subscription information in persistent storage, such that subscriptions survive power cycles and Boot Manager resets. A client can subscribe with the SA for Trap 64/65 ‘Port In/Out of Service’ to detect when the node on which the Boot Manager resides is reset. The client can also subscribe with the Boot Manager for the Heartbeat notice (see A5.6.7.3.3 "Heartbeat" on page 1350), so it can detect when the Boot Manager disappears or ceases to function. Successful reception of the heartbeat indicates that the Boot Manager is alive and that the client’s subscriptions are still valid.
- Boot Manager failover - In the event that the active Boot Manager fails and another Boot Manager takes over, subscriptions with the old Boot Manager might not be carried over to the new Boot Manager. For graceful failover, the active Boot Manager provides subscription information to standby Boot Managers before replying to a subscription request. A client can subscribe with the SA for Trap 65 ‘Port Out of Service’ to detect when the node on which the Boot Manager resides goes down and subsequently query the SA to locate the new Boot Manager. If the client subscribed with the Boot Manager for the Heartbeat notice (see A5.6.7.3.3 "Heartbeat" on page 1350), the heartbeat can indicate graceful failover. Successful reception of the heartbeat indicates that the new Boot Manager inherited the client’s subscriptions.
- Temporary path disruptions can make a client unreachable, which might result in the Boot Manager timing-out and removing the client’s subscriptions. Both the client and the manager can subscribe to SA trap 65 to detect such a condition (assuming that the SM detects the path failure). Again, the client can subscribe with the Boot Manager for the Heartbeat notice (see A5.6.7.3.3 "Heartbeat" on page 1350), so it can detect when the Boot Manager drops its subscriptions. Successful reception of the heartbeat indicates that the client’s subscriptions are still valid.

oA5-67.2.6: If the Boot Manager supports Persistent Context (i.e. sets the *IsContextPersistent* bit in BootMgtGetResp(ClassPortInfo) to 1), it shall

retain subscriptions across power cycles (i.e., use subscription information stored in non-volatile storage).

oA5-67.2.7: If the Boot Manager supports Persistent Context (i.e. sets the *IsContextPersistent* bit in BootMgtGetResp(ClassPortInfo) to 1), it shall save subscription information in non-volatile storage before responding to the BootMgtSet(InformInfo).

oA5-67.2.8: If the Boot Manager supports Graceful Failover (i.e., sets the *GracefulFailover* bit in BootMgtGetResp(ClassPortInfo) to 1), it shall share subscription information with all standby Boot Managers provided by that same vendor. This includes supplying Standby Boot Managers with subscription information when the standby manager comes on-line and updating standby Boot Managers when subscription information changes before responding to the BootMgtSet(InformInfo).

oA5-67.2.9: If the Boot Manager supports Graceful Failover (i.e., sets the *GracefulFailover* bit in BootMgtGetResp(ClassPortInfo) to 1), it shall make subscription information known to all standby Boot Managers provided by that same vendor before responding to the BootMgtSet(InformInfo).

A5.6.7.3.2 SUBSCRIPTION TIMEOUT

When a subscribed node fails or resets, it does not always have the chance to unsubscribe. For the express purpose of limiting the number of retries and the size of the Boot Manager's subscribers list over long periods of time, clients that become unreachable by the Boot Manager or otherwise leave the fabric for any reason will have their subscription terminated by the Boot Manager. However the Boot Manager needs to be able to distinguish between subscribers that are temporarily unreachable verses those that have gone away.

When the Boot Manager receives a trap, it sends a BootMgtReport() to each subscriber and waits for a BootMgtReportResp(). If it fails to receive the response, it resends the BootMgtReport(). The minimum amount of time the Boot Manager waits before re-sending is specified in the InformInfo attribute. The Boot Manager should implement the Retry-Backoff Policy specified in Annex A1 section A1.3.2 between successive attempts to limit the number of retries when trying to reach an unreachable node. If after 10 minutes the subscriber has not responded, the Boot Manager may terminate the retry and cancel all of the subscriptions for that client.

oA5-67.2.10: If the Boot Manager fails to receive a ReportResp() in response to a Report(), then the Boot Manager shall continue to retry the Report() until either it receives a ReportResp(), the client unsubscribes, or the Boot Manager terminates the subscription.

oA5-67.2.11: The Boot Manager shall not terminate a subscription due to a missing ReportResp() until after 10 minutes of retrying has occurred. If the Boot Manager does terminate a subscription due to a missing ReportResp(), it shall terminate all subscriptions for that particular destination (LID/GID + QPN).

A5.6.7.3.3 HEARTBEAT

The Boot Manager periodically sends a BootMgtReport(Notice=Heartbeat) to let subscribers know that the Boot Manager still exists and that the client's subscriptions are still valid. It is also a means for the Boot Manager to detect stale subscriptions since failure for a client to respond to the heartbeat is grounds to cancel that client's subscriptions.

When a client subscribes for the heartbeat event, the Boot Manager sends the client a BootMgtReport(Notice=Heartbeat) before it replies to the subscription request. This initial heartbeat provides the client with the maximum time until the next heartbeat (TTNH). The length of time between heartbeats is a Boot Manager policy. The client starts an expiration timer for TTNH minutes and if the timer expires, it is an indication that the Boot Manager is no longer functional or that the Boot Manager has cancelled the client's subscriptions. Thus, the client should re-subscribe or query the SA to locate the new Boot Manager so it can subscribe. Each time a client receives a BootMgtReport(Notice=Heartbeat), it resets its expiration timer to the new TTNH value specified in the Heartbeat notice.

oA5-67.2.12: For heartbeat subscriptions, the Boot Manager shall generate a BootMgtReport(Notice=**Heartbeat**) to the subscriber before sending the BootMgtGetResp(InformInfo)

oA5-67.2.13: The Boot Manager shall continue to generate BootMgtReport(Notice=**Heartbeat**) to each subscriber within TTNH minutes after sending the previous BootMgtReport(Notice=Heartbeat) to that subscriber.

Note that the LID Range and GID in the InformInfo for a heartbeat report is irrelevant and thus ignored by the Boot Manager. That is, the Boot Manager generates a Heartbeat report to anyone subscribed for heartbeats regardless of the GID and LID range in the InformInfo attribute.

The Heartbeat notice also has a 'Fail-Over' bit that indicates a new manager. This is to inform the client that a standby manager has gracefully taken over the Boot Manager responsibility. In order to ensure that the notice came from a valid Boot Manager, instead of using the Report()'s reciprocal path to communicate with the new Boot Manager, the client should query the SA to locate the new Boot Manager, but it will not need to re-subscribe.

oA5-67.2.14: The Boot Manager shall not set the FailOver bit in a Heartbeat notice unless the Boot Manager inherited all of the subscriptions from the previous manager.

oA5-67.2.15: The Boot Manager shall only set the FailOver bit in the first Heartbeat notice it sends to a client. Thus, the Boot Manager shall not set the FailOver bit in subsequent reports to a client after it receives the ReportResp() from that client.

A5.6.8 INFORMINFO ATTRIBUTE

The InformInfo attribute provides information for subscribing to a class manager for event forwarding. See 13.4.8.3 InformInfo on page 739 and 13.4.11 Event Forwarding on page 745.

Table 385 InformInfo

Component	Access	Length (bits)	Offset (bits)	Description
See InformInfo in 13.4.8.3 InformInfo on page 739				

A5.7 PLATFORMS USE OF BIS

A5.7.1 BIS USAGE OVERVIEW

This section describes the behavior of a booting platform that supports BIS as a Boot Resolution Method. BIS is described in the Boot Information Service Annex. A booting platform may support using BIS, Persistent storage, or both as the source for boot information. The overview and operation of a BIS server is described in General Operation in the Boot Information Service Annex.

The BIS server returns many of the same attributes to the booting platform as the BootManager provides.

The booting platform can access a BIS without requiring a BtA to also be running on the booting platform. Ideally, a booting platform accessing a BIS should also run the BtA so that the BootManager can receive Traps, read the Notice Queue or read status and status components using PlatformBootInfo to stay informed of BIS related events.

The booting platform queries a BIS using the BIS class MADs as described in the BIS Annex. The booting platform queries a BIS using the BisQuery(BootQueryInfo) to find information records (PlatformBootInfo and PortBootInfo) and locator records (RomRepositoryLocatorRecords, ConsoleLocatorRecords, and OsLocatorRecords). The BIS returns the

requested attributes using BisQueryResp(*). The BisQuery(*) and BisQueryResp(*) use the Reliable Multi-Packet Protocol.

oA5-68: If BIS is supported, the platform shall conform to the BIS Class MADs and data structure as specified in the BIS Annex.

oA5-69: If BIS is supported and the platform queries the BIS for locator records, then the platform shall use the locator records (RomRepository-LocatorRecord, ConsoleLocatorRecord, and OsLocatorRecord) in the order in which they were received from the BIS.

A booting platform that supports BIS but not the BtA defaults to using the BIS to acquire the PlatformBootInfo and PortBootInfo attributes. If the platform contains a BtA, then the BootManager sets these attributes and disables the platform from acquiring them from the BIS.

A5.7.2 PLATFORMBOOTINFO SOURCE

If BIS or Persistent boot methods are supported, then the booting platform must determine the source of the platforms booting parameters. The factory default for a booting platform is to use a BIS to obtain the PlatformBootInfo attribute.

If the platform has a BtA, the default usage of the BIS may be altered by the BootManager, depending on the facilities booting policy, by modifying the PlatformBootInfoSource and PortBootInfoSource components in the PlatformBootInfo attribute.

The booting platform tests PlatformBootInfoSource (see Table 370 on page 1302) to determine the source of the PlatformBootInfo attribute.

oA5-70: If BIS is supported and PlatformBootInfoSource=0, then the booting platform shall query the BIS for the PlatformBootInfo attribute.

oA5-71: When the booting platform supports both BIS and local persistent storage, if the local PlatformBootInfoSource=1, then the booting platform shall use the PlatformBootInfo attribute in persistent storage.

Before the booting platform first accesses a BIS, the platform might not contain information regarding the subnet specific boot time-out values. The booting platform must use the default values until the BIS returns the PortBootInfo attribute.

CA5-40: If the booting platform supports BIS and the platform does not have a persistent PlatformBootInfo attribute, then the booting platform shall use the default values for PlatformBootInfo until it queries the BIS for PlatformBootInfo.

A5.7.3 PORTBOOTINFO SOURCE

If BIS or Persistent boot methods are supported, then the booting platform must determine the source of subnet parameters, such as time-outs located in the PortBootInfo attribute. The factory default for a booting platform is to use a BIS to obtain PortBootInfo attributes.

The booting platform tests PortBootInfoSource (see Table 370 on page 1302) to determine the source of the PortBootInfo attribute. If PortBootInfoSource=0 (the default value), then the booting platform shall query the BIS for PortBootInfo. If PlatformBootInfoSource=1, then the booting platform shall use the PortBootInfo attribute in persistent storage.

oA5-72: If BIS is supported and PortBootInfoSource=0, then the booting platform shall use the BIS as the source of the PortBootInfo attribute.

oA5-73: If PortBootInfoSource=1, then the booting platform shall use persistent storage as the source of the PortBootInfo attribute.

When the booting platform first accesses a BIS, the platform might not contain information regarding the subnet specific boot time-out values. The booting platform must use the default values until the BIS returns the PortBootInfo attribute.

oA5-74: If BIS is supported and PortBootInfoSource=0, then PortBootInfo:BisTimeout, PortBootInfo:EndnodeTimeout, and PortBootInfo:InitTimeout components default values specified in Table 372 "PortBootInfo Attribute" on page 1321 shall be used.

A5.7.4 DETERMINING TO USE A BIS

A booting platform shall use the BIS as a source of booting information if:

- PlatformBootInfoSource=1 or
- PortBootInfoSource=1 or
- Any Locator Source in PlatformBootInfo in persistent storage indicates BIS

oA5-75: If BIS is supported, the booting platform shall only use PlatformBootInfoSource, PortBootInfoSource, RomRepositoryLocatorSource, ConsoleLocatorSource, and OsLocatorSource in the PlatformBootInfo attribute to determine if the BIS is used as a source for booting information.

A5.7.5 FINDING A BIS

Once the platform determines that it needs a BIS, the platform determines the location of the BIS using the service advertisement in the SA (see A6.3.1 Registration on page 1411). The SA is used to find the ServiceRecord of a BIS for the subnet. From the ServiceRecord, the BIS Ser-

viceGID and partition is found. A PathRecord can be obtained from the SA using the GID and partition from the ServiceRecord. Using the PathRecord the BIS service is located on QP1 or redirected to another QP. This process should be followed each time the platform boots to prevent the platforms BIS Service Record from becoming stale.

oA5-76: If BIS is supported and the platform determines that the BIS will be queried, then the booting platform shall:

- locate the BIS using SubnAdmGet(ServiceRecord) using only the ServiceName of “BIS.IBTA” as described in the BIS Annex.
- fill out the PathRecord request specifying GID and partition returned in the SubnAdmGet(ServiceRecord) response from the SA.

A5.7.6 SELECTING A BIS

Once the platform determines that a BIS is to serve as a source of booting information, then the platform determines which BIS to access. If the booting platform is on a single subnet with multiple BISs or if the platform is on multiple subnets with each subnet containing a BIS service, then the platform selects a BIS to use as the source of booting information.

Any BIS can supply a platform with attributes that have platform scope (i.e., PlatformBootInfo, RomRepositoryLocatorRecords, ConsoleLocatorRecords and OsLocatorRecords attributes).

A single BIS can also provide a platform with port specific attributes (PortBootInfo) for each of its ports when responding to a BootQueryInfo:PortGUID with BootInfoRequested=0x01. However, a BIS might not have port information for all ports of the booting platform, and thus, the booting platform may need to query another BIS (e.g., a BIS on the same subnet as the port).

If a subnet contains multiple BISs as indicated by multiple records being returned in the SubnAdmGet(ServiceRecord) response, then the platform queries each BIS in order until the requested information is returned.

A5.7.7 PRIORITIZING MULTIPLE BISS

When multiple BISs exist in a subnet or a platform is on multiple subnets each having some number of BISs, the booting platform should determine which BIS to use as its source of booting information. The booting platform can query the SA searching for a BIS in the BisPortPriority order.

If multiple records are returned in a SubnAdmGet(ServiceRecord) response then multiple BISs on that port serve the booting platform.

oA5-77: If BIS is supported, then the booting platform shall prioritize multiple BISs by using records returned by a BIS in BisPortPriority order from highest to lowest.

oA5-78: If a subnet has multiple BIS's then the platform will access the BIS's in the order in which the ServiceRecords are return from the SA.

If a BIS does not provide all of the records needed by a booting platform then other BIS's are queried to obtain the desired record(s).

A5.7.8 OTHER CONSIDERATIONS

oA5-79: If BIS is used to locate a ROM Repository, Console, or OS Loader, then the platform shall only query the BIS for records that the platform supports.

oA5-80: If BisQueryResp() returns PortBootInfo:NetworkBootPortPriority = 0x0 then the platform shall not use this port for IB network booting.

oA5-81: If BisQueryResp() returns PortBootInfo:RomPortPriority = 0x0 then the booting platform shall not attempt to access the RomRepository through this port.

oA5-82: If BisQueryResp() returns PortBootInfo:ConsolePortPriority = 0x0 then the booting platform shall not attempt to access the IB Console through this port.

oA5-83: If BisQueryResp() returns PortBootInfo:IocPortPriority = 0x0 then the booting platform shall not attempt to boot through this port.

oA5-84: If the platform supports BIS, the BtA, and Traps, then the BtA shall issue a StatusReport Trap (see Table 382 "Notice Details for Trap 0x0110 - StatusReport" on page 1343) to every port for failures defined in the notice details.

oA5-85: If the platform supports BIS, the BtA, and notice queue, then the BtA shall post a StatusReport Notice (see Table 382 "Notice Details for Trap 0x0110 - StatusReport" on page 1343) to the notice queue for failures defined in the notice details.

A5.7.8.1 RELIABLE MULTI-PACKET PROTOCOL

oA5-86: If the platform supports BIS, then the platform shall support the Reliable Multi-Packet Protocol specified in 13.6 Reliable Multi-Packet Transaction Protocol on page 770.

A5.7.8.2 PORT GID TO LID RESOLUTION

Given the PortGID from the Locator Record, the booting platform queries the SA with SubnAdmGetTable(PathRecord). The booting platform que-

ries the SA to obtain a path to the IOC in the Locator Record. A technique that will find all possible paths, in case some are inoperable, is to fill out the PathRecord request specifying only the SGID of the booting platform, the DGID equal to the portGID from the Locator Record, and all other components “wildcarded” (ComponentMask bits set to 0) so they match anything. Note, the SLID is “wildcarded” to account for a possible non-zero LMC. A table of paths are returned for all paths from the booting platform portGID to the portGID specified in the Locator Record.

A5.7.8.3 REPORTING FAILURES

A platform that supports a BootAgent and Traps has the ability to notify the BootManager when unauthorized accesses occur within both the Boot Management and BIS class MADs. This allows reporting of the unauthorized attempt to manipulate the booting platform so that a higher level management entity (Administrator) can take an appropriate action.

A5.8 IB NETWORK BOOTING

A boot platform may be capable of booting over IB using the IB Network boot described in Figure 272 ”IB Network Boot Model” on page 1279 without requiring a LAN NIC. This Annex requires that a booting platform make the BootManager aware of the capability of using the IB Network boot method by setting Network Boot-IB Network Model in PlatformBootInfo:Capability in Table 370 on page 1302).

There are multiple network protocols that allow a booting platform to directly communicate with a boot server such as IP over IB (IPoIB) and Sockets Direct Protocol (SDP). OsLocatorRecords inform the booting platform when to attempt IB Network Boot and which IB network protocol to use. The OsLocatorRecord also has the ability to identify the boot server and to supply protocol specific parameters need to perform the IB Network Boot.

A booting platform may contain multiple ports on more than one subnet. Instead of enumerating each port, the BootManager may inform the booting platform of the port order in which to attempt to find the IB Network boot server by setting NetworkBootPortPriority in PortBootInfo (see Table 372 on page 1321)

A5.9 RETRY BACKOFF

When many platforms in a subnet are booting at the same time, then congestion on some resources while booting may develop. A1.3.2 Retry-Backoff Policy on page 1134 contains guidance for IB retries.

CA5-41: A booting platform shall implement the Retry Back-off Policy specified in the I/O Annex when it fails to receive a response to any MAD for any management class.

A5.10 IB BOOT PROCESS - SUMMARY

Boot Management MADs specify messages that appear on the wire and behaviors associated with those messages. The appearance of a message at a port implies a required action and, most likely, a response. Additionally, the appearance of a message on the wire implies behavior of the entity that caused the message to be emitted. Various conceptualizations are used in specifying behaviors of the booting platform. However, the use of such conceptualizations in this Annex is purely a descriptive artifice. The conceptualizations themselves, do not convey normative requirements. While some conceptualizations used in the this Annex may suggest certain implementations, implementations are outside of the scope of this Annex and no specific implementation is implied.

Booting over IB involves selecting Locator Records to locate a ROM Repository, a Console object, and/or the OS boot loader. There are 2 sources of Locator Records, Local-Persistent and BIS. Local-Persistent Locator Records may be set using Boot Management. BIS is an optional service on the IB fabric. Each time the platform boots, it uses either Local-Persistent records or queries the BIS, which returns the latest set of Locator Records for the booting platform to use.

Booting over IB includes the platform determining if the boot environment will be extended and if device drivers will be loaded using the ROM Repository. Booting also includes determining if an IB console will be used and selecting the device or network to be used if booting over IB.

Booting platforms which support a BtA may contain non-volatile storage. Booting platforms can optionally permit the system administrator, via the BootManager, to update attributes that affect booting. Platforms may implement a local only update policy that does not allow updates using IBA methods. If the local policy allows a BootManager to write components, then the R/W components of BootMgt attributes are stored in the platforms non-volatile storage. A ladder diagram of a Boot Manager configuring the boot policy of a boot platform is shown in Figure 278. The boot platform in this example supports an extension of the boot environment, persistent boot, ROM locator record and OS locator record updates.

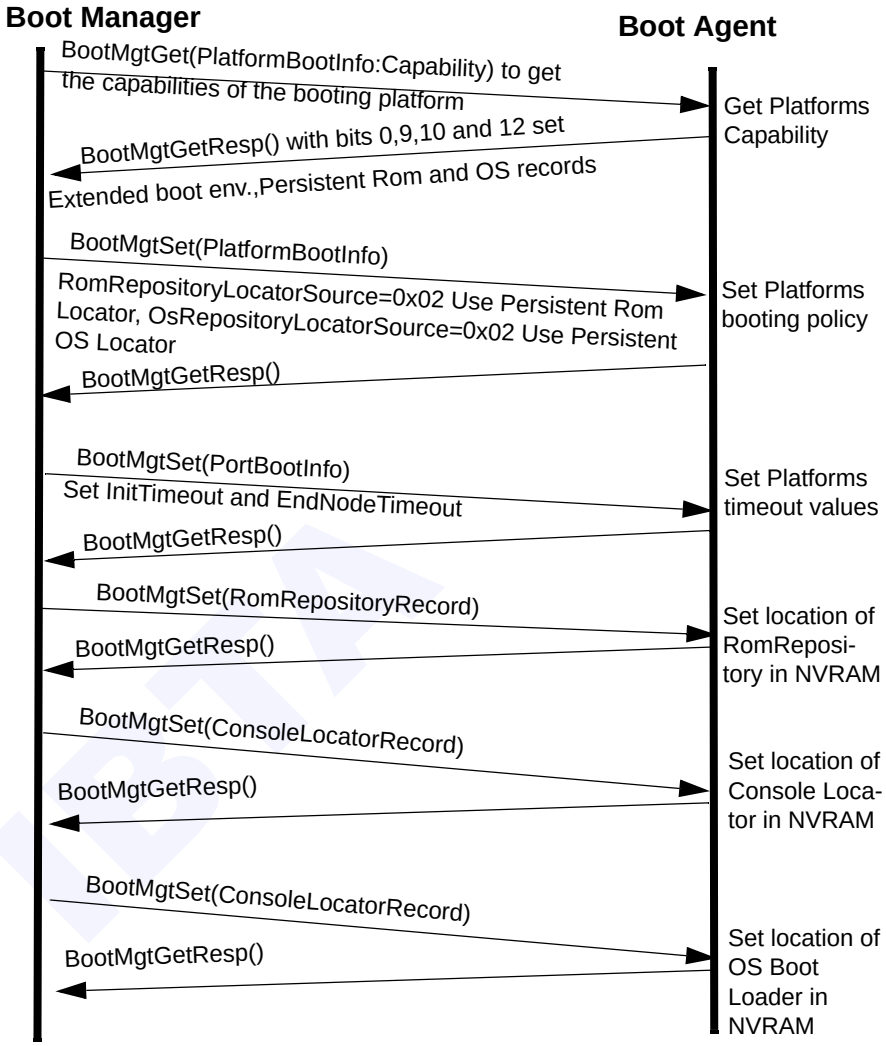


Figure 278 Configuring the Boot Platform

A platform with invalid data in this non-volatile storage (NVRAM) might be a platform shipped directly from the factory, a platform which has had its entire non-volatile storage cleared, or a platform that does not contain valid data in non-volatile storage. When a platform boots with invalid Boot Management data in non-volatile storage then:

1) If the platform supports Local-Persistent Locator Records, then the booting platform may wait for the BootManager to Set() Persistent Locator Records to the platform. After being Set() the platform will use the Persistent Locator Records to optionally load the boot environment extension, device drivers, the Console, and the OS Boot Loader.

2) If the platform supports BIS, then it may use BIS Class MADs to query the BIS. The BIS will, depending on the query, return Locator Records for the boot environment extension, the Console, and the OS Boot Loader. Proprietary Device Drivers that are not co-located with the boot environment may be found in a ROM Repository image.

Other considerations are:

1) A booting platform queries the SA using SubnAdmGet(ServiceRecord) to locate a BIS. The PortBootInfo(BisTimeout) value determines the minimum amount of time the booting platform should wait for the BIS to register with the SA. If BisTimeout has not been set, then the default value listed in the component definition should be used.

2) If the booting platform supports both Local-Persistent and BIS boot resolution methods, then the platform defaults to BIS only until the Boot Management attributes are written.

3) The booting platform may optionally extend the boot environment.

4) The booting platform may optionally locate the Console.

5) The booting platform may optionally locate the OS Boot Loader. This is optional because the BtA may be on the platform to extend the boot environment or use the IB console, but have no need to boot over IB.

If the booting platform contains valid persistent data in non-volatile storage, then the platform does not wait for the BootManager to Set() boot attribute to the platform. Here, the platform will either use Local-Persistent and/or BIS Locator Records to locate the extended boot environment code, the Console, or the OS Boot Loader.

Figure 280 on page 1361 describes a platform boot example that illustrates the steps involved in booting a platform. The boxes on the left side of the page show that a booting platform would access the ROM Repository, Console, and OS loader. The single box on the right shows the common steps used by the booting platform each time the platform needs to determine the location of a device or service. When the platform is powered on, reset, or rebooted, then the platform finds devices or services needed to boot. In this example, the ROM Repository images are accessed first. Then the console is located followed by the OS boot loader.

The order (ROM Repository, Console, and OS loader) is determined by the booting platform. Platforms may already contain a local copy of a console device driver used to access the console, which enables that platform to immediately connect to the console.

ROM Repository

1) The Boot Resolution Method for the ROM Repository is selected using ROMRepositoryLocatorSource.

   a) The ROMRepositoryLocatorSource selects BIS or BIS then Local-Persistent or Local-Persistent then BIS or none. This variability is not shown in Figure 280 ”Platform Boot Example” on page 1361 and with the intent being if one method fails the secondary method may be used.

2) 1) above provides the booting platform with one or more RomRepositoryLocatorRecords that point to the IOU.

   Using the Locator Record the platform queries the SA for a path and connects to the ROM Repository. Connecting to the RomRepository is shown in Figure 279. The ROM Repository protocol is detailed in A5.12 "ROM Repository" on page 1365.

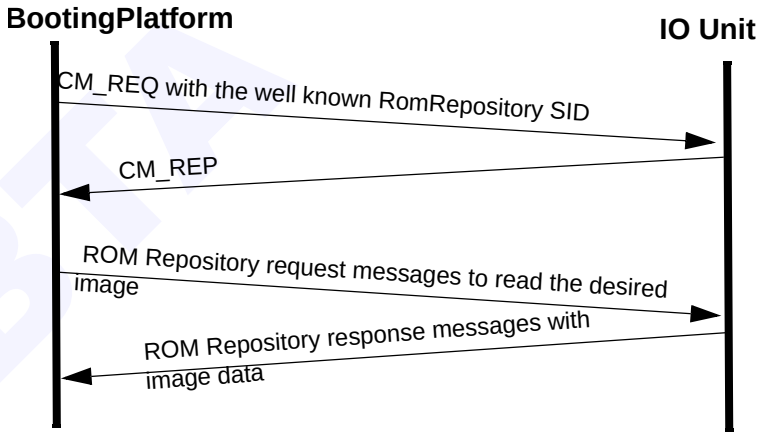


Figure 279 RomRepository Access

3) The platform then searches the ROM Repository for the desired image(s).

4) Failures throughout the process should result in Traps being issued (see A5.6.7 "Traps and Notice Queues" on page 1336).

Console

The Boot Resolution Method for the Console is similar to the ROM Repository except that the method is selected using ConsoleLocatorSource and the Console protocol described in Annex A2 Console Service Protocol or a proprietary console protocol is used.

OS Loader

The Boot Resolution Method for the OS Loader is similar to the ROM Repository and Console except that the method is selected using OsLocatorSource and the protocol is I/O specific (e.g. SRP, IPoIB and proprietary).

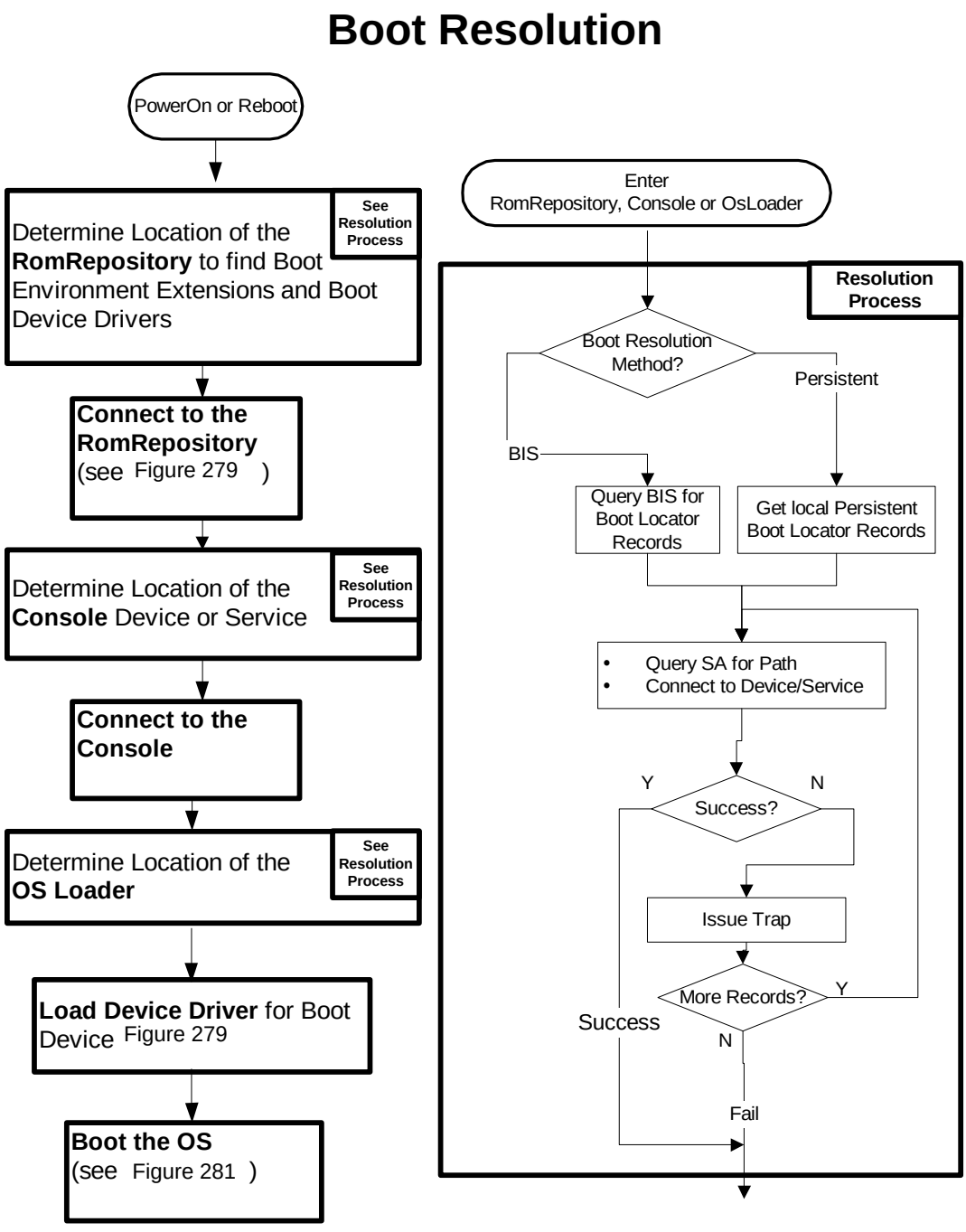


Figure 280 Platform Boot Example

Figure 281 is a ladder diagram illustrating the messages a booting platform sends to an IOU and the IOU response. The booting platform obtains

information about the IOC, I/O protocol and boot device from the Locator Record stored in the persistent storage or obtained from the BIS.

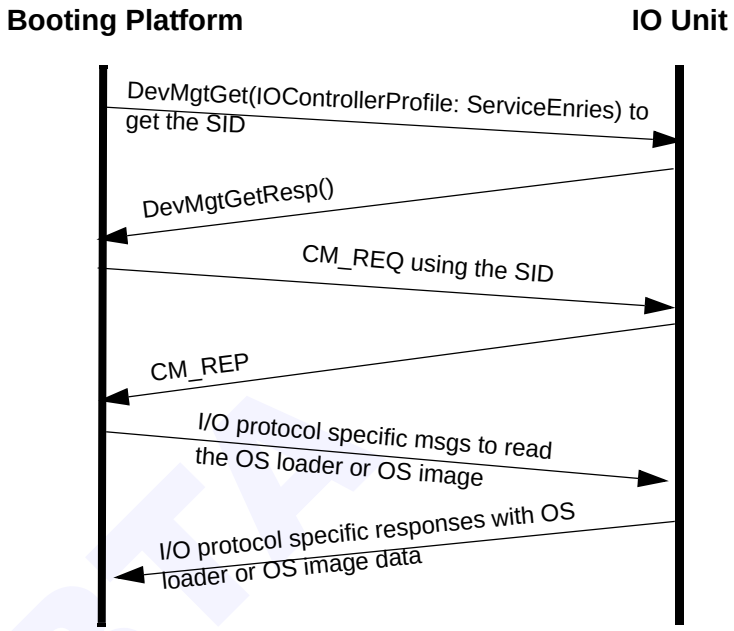


Figure 281 Boot Platform to IOU Communication - Access OS Loader

A5.11 ADDITIONALINFO

AdditionalInfo is a component of the ConsoleLocatorRecord and OsLocatorRecord attributes provided to a booting platform by the BIS or the BootManager.

AdditionalInfo is a 160 byte field whose content is dependent on the DeviceType, ServiceType, and I/O protocol. This information is intended to be passed to the device driver. This component provides information that might help the I/O driver configure itself, configure the IOC, and/or identify a particular I/O device or service.

The contents of the AdditionalInfo component contain multiple variable length elements. Each element contains 3 fields. The first field, ***Type*** is a one byte field that identifies the content of the element. The second field, Length is a one byte field and specifies the length of the Value field. The third field, ***Value*** contains the 'Length' number of bytes of element data. There may be any number of elements limited only by the 160 byte size of the overall AdditionalInfo component. The remainder of the AdditionalInfo component is filled with 0x00.

Table 386 General AdditionalInfo Element Definitions

Type Field Value	Length Field Value	Description
0x00	0x00	Marks end of elements - ignore all data following this value
0x01	variable	ProtocolName in ASCII (UTF-8) - precedes any protocol specific components
0x02	variable	Filename - in ASCII (UTF-8)
0x10 - 0x1F	variable	Device Vendor Specific
0x20-0x3F	variable	Protocol Specific - (see following sections for examples)

This annex describes AdditionalInfo elements for the IBA Console and SRP protocols.

For an IOC that supports multiple protocols, the AdditionalInfo component can contain multiple sets of protocol specific elements. All elements be tween two ProtocolName elements are for the protocol identified by the preceding ProtocolName element. Thus, each ProtocolName element identifies a new group of protocol specific elements.

A5.11.1 SRP

When using the SRP protocol, additional information is passed to the entity on the booting platform (i.e. boot environment code) that loads the OS Loader from a storage device into memory through the OsLocatorRecord attribute provided by either the BIS or the BootManager. Table 387 specifies elements specific to the SCSI RDMA Protocol.

Table 387 SRP AdditionalInfo Elements

Type Field Value	Length Field Value (Bytes)	Description
0x01	7	ProtocolName = "**SRP.T10**" in ASCII.
0x20	16	SCSI Initiator Port Identifier - see IB annex of the SRP specification for format.
0x21	16	SCSI Target Port Identifier - see IB annex of the SRP specification for format.
0x22	8 or 16	Logical Unit Name - format type either: • NAA IEEE Registered - 8 bytes • NAA IEEE Registered Extended - 16 bytes

For an IOC that supports multiple protocols, the AdditionalInfo component can contain multiple sets of protocol specific elements. All elements be-

tween two ProtocolName elements are for the protocol identified by the preceding ProtocolName element. Thus, each ProtocolName element identifies a new group of protocol specific elements.

The ProtocolName for SRP is identical to the ServiceName defined by T10 for SRP. Value types 0x20 to 0x3F are protocol specific and depend on the ProtocolName that proceeds them.

The SCSI Initiator Port Identifier type element specifies the SRP initiator port identifier value that the booting platform uses to access the SPR target.

The SCSI Target Port Identifier identifies the SRP target. A OsLocator-Record should be associated with one SRP target.

The Logical Unit Name is the name returned in the response to SCSI Inquiry Vital Product Date Page Code 0x83, the Device Identification Page. This name uniquely identifies the Logical Unit from every other one in the world. The booting platform uses the Logical Unit Name to determine which Logical Unit Number (LUN) it uses to access the boot loader. A OsLocatorRecord should specify one Logical Unit Name in the AdditionalInfo component. A booting platform that obtains a OsLocatorRecord for an SRP target should:

- Check the 0x01 type element for a content match with the SRP Service-Name of “SRP.T10”
- Use the 0x20 type element as the SCSI Initiator Port Identifier.
- Establish an SRP connection with the IOC indicated in the OsLocatorRecord.
- Use the 64 bit Target port extension identifier portion of the 0x21 type element to match the 64 bit extension identifier portion of the ServiceName from the Service Entries component of IOControllerProfile attribute. Use the ServiceID corresponding to the matching ServiceName.
- Retrieve LUN information with the SCSI Report LUNs command.
- Query each LUN with SCSI Inquiry Vital Product Data Page Code 0x83 until a response with a Logical Unit Name exactly matches the 0x22 type element. This is the boot LUN. If any of these steps fail then disregard the OsLocatorRecord.

How the BootManager or the BIS obtains information contained in AdditionalInfo is out of the scope of this Annex.

A5.11.2 CONSOLE

Additional information is intended to be passed to the entity on the booting platform (i.e. boot environment code) that is communicating with the IB

CSP Server. Table 388 specifies elements specific to the IB Console Service Protocol.

Table 388 Console AdditionalInfo Elements

Type Field Value	Length (160 Bytes)	Description
0x01	0x0C	ProtocolName = “Console.IBTA” in ASCII
0x20	0x08	ResumeKey - Key used in SessionResumeRequest
0x21	0x04	DeviceNum - Requested DeviceNumber
0x22	0x04	ConsoleUserID - Identifier of the console user at the CSP client requesting the session
0x23	0x08	Nickname - Short, null terminated ASCII string identifying the console user
0x24	0x40	ConsoleUsername - Long, null terminated ASCII string identifying the console user
0x25	0x0C	Capability - This element contains the ConsoleCapabilityRecord as specified in the Console Annex and informs the Booting platform of the capabilities of the console service pointed to by the Console locator record.

For a console that supports multiple protocols, the AdditionalInfo component can contain multiple sets of protocol specific elements. All elements between two ProtocolName elements are for the protocol identified by the preceding ProtocolName element. Thus, each ProtocolName element identifies a new group of protocol specific elements.

A5.12 ROM REPOSITORY

A5.12.1 INTRODUCTION

IBA provides an optional method for managed I/O units to supply code and/or data images to a node by supporting a ROM repository. This section describes the wire protocol for a node to access a managed I/O unit’s ROM repository. The ROM repository protocol provides a method for a node to search a managed I/O unit for code images needed by the node and download them from the repository. The protocol also provides a method for a node to add new images to the repository, delete images from the repository, and to modify existing images in the repository. This section focuses on ROM repository containing boot driver images. Other technologies provide similar functionality through the use of Option ROMs within adapter cards. It is expected that this repository survives power cycling much like Read-Only-Memory (ROM) devices, hence the term ROM Repository.

A5.12.2 OVERVIEW OF IOC BOOT DRIVER DOWNLOAD

The following are the normal steps in booting a platform using a driver supplied by a managed I/O unit. The term platform refers to the combination of processes and utilities being performed regardless of whether they are provided by the CA, a boot environment such as BIOS or IEEE 1275, the platform vendor, or the IO driver.

1) The booting platform selects a boot IOC using either persistent or BIS type boot records (see A5.6.3.4 "Boot Record Locator Sources" on page 1315).
2) The booting platform sends DevMgtGet(IOControllerProfile) to the I/O unit that contains the selected IOC and then creates compatibility strings using the components returned in the DevMgtGetResp(IO-ControllerProfile) as specified in section G.1.3.1.1 of the I/O Annex.
3) The booting platform selects a managed I/O unit that provides a ROM repository. This might be the I/O unit containing the boot IOC or it might be an I/O unit identified using one of the resolution methods specified in A5.6.3.4 "Boot Record Locator Sources" on page 1315.
4) The booting platform searches the selected IOU's ROM repository for ROM image that matches selected IO controller and type of boot environment.
5) The booting platform loads boot driver image into its boot environment.
6) Boot environment executes the boot driver image.
7) Boot driver image calls IBA services to:
   a) create QPs
   b) form connections
   c) invoke IBA transport (Send/Receive/RDMA)

A5.12.3 ROM REPOSITORY MODEL

The ROM repository protocol supports multiple concurrent connections to a ROM repository. The number of connections supported is an implementation option which can be as few as one (i.e. only one connection to the ROM repository at any given time). Multiple connection support is desirable where many nodes depend on a single IOU's ROM repository.

The Capabilities component of RepositoryInfoResponse message (see Table 391 on page 1370) specifies whether or not the IOU supports multiple connections.

The main objective of defining the ROM repository is for providing IOC drivers. IBA defines the ROM repository space that contains IOC driver

images (Option ROM page). This section primarily deals with the ROM repository space that contains IOC driver images.

IBA does not limit the type of data that is located in the ROM repository. An IOU may support more than one type of repository space and each repository space has a unique “ROM repository ID” (see Table 394 on page 1375). Other ROM repository spaces for example, might contain microcode for an IOC or tabular data. These ROM repository spaces are not described in this annex, but are left to innovation. Each ROM repository must have an Option ROM page, however, the number of images in that page may be zero.

Each image in the Option ROM page is described by an Image descriptor (see Table 395 on page 1376). The information contained in an Image descriptor is used to locate an IOC driver image that matches to a particular hardware defined by the IOControllerProfile attribute. A requestor uses compatibility string(s) (Ref. I/O Annex Section A1.2.4) specified in the image descriptor to match the image. If there is a common driver image for multiple types of IOCs, then the image descriptor contains multiple compatibility strings to match the image. Image descriptor size is variable and depends on the number of compatibility strings specified in the descriptor.

Each image in the Option ROM page has a unique handle assigned by the IOU. The image handle is specified in the image descriptor and requestors use the handle to access the image.

It is not required that a ROM image be from the same IO unit that contains the IO controller. In fact, it is desirable to permit a single ROM image to drive multiple IO controllers, on various IO units. Thus, if the target IO unit does not provide a driver, the host may search for other ROM repositories that contain a suitable driver. In fact, the host policy may be to discover all ROM repositories and then select the best match between all of the option ROM images and the target IO controller. Console and OS Locator Records contain a DeviceDriverLocation component as a hint of which repository most likely contains the best driver for a particular I/O controller.

A5.12.4 IDENTIFYING A ROM REPOSITORY

ROM repository support by a managed I/O unit is optional. An IOU indicates that it contains a ROM Repository by setting the Option ROM component in IOUnitInfo (device management) attribute. A host reads the IOUnitInfo attribute by sending a DevMgtGet(IOUnitInfo). The IO unit responds with DevMgtGetResp(IOUnitInfo). If the Option ROM bit is set, it means that a ROM repository exists on the IOU and the IOU has at least one QP for the ROM repository protocol so that a requestor can establish a connection with the IO unit and access the ROM repository.

CA5-42: An IO unit that supports a ROM repository shall set the option ROM bit in its IOUnitInfo attribute.

A5.12.5 ROM REPOSITORY ACCESS METHODS

ROM Repository messages use RC transport services. The requestor sends a connection request (see Volume 1, Chapter 12) specifying the well-known option ROM Service ID 0x0000_0000_0000_0001, defined in the “IB Identifiers” Annex, requesting a reliable connection. An IOU may use any of its available queue pairs to satisfy the connection request. Once the connection is established, the requestor uses the ROM repository protocol to read, write, modify, and delete an image in the ROM repository.

ROM repository access messages are relayed strictly by Send operations (RDMA operations are not used in either direction). Access message formats (see Table 389 on page 1368) dictate that payload byte zero indicates the message type, the contents of the remainder of the message are dictated by that message type.

An IOU may allow multiple simultaneous connections using the well-known SID. An IOU with multiple connection support (see Capabilities component in Table 391 on page 1370) can allow simultaneous reading of one or more images by multiple requestors.

CA5-43: An IO unit that contains a ROM repository shall provide at least one QP supporting a reliable connection for ROM repository access.

CA5-44: An IO unit that supports ROM repository shall only accept send messages on its ROM repository QPs.

The ROM repository protocol relies on message-level flow control supported by the hardware.

A5.12.6 ROM REPOSITORY MESSAGES

ROM repository operation codes and their corresponding message names are as specified in Table 389 on page 1368.

Table 389 ROM Repository Operation Codes

OpCode	Message Name	Description
0x00	RepositoryInfoRequest	Request information on the ROM repository from the IOU. See Table 390 "RepositoryInfoRequest Message" on page 1370.

Table 389 ROM Repository Operation Codes (Continued)

OpCode	Message Name	Description
0x01	DescriptorReadRequest	Request one or more image descriptors from the IOU's ROM repository. See Table 397 "DescriptorReadRequest message" on page 1379.
0x02	ImageReadRequest	Request image data from an IOU. See Table 399 "ImageReadRequest Message" on page 1381.
0x03	ImageAddRequest	Initiate an image add operation. See Table 401 "ImageAddRequest Message" on page 1385.
0x04	ImageUpdateRequest	Initiate an image update operation. See Table 403 "ImageUpdateRequest Message" on page 1389.
0x05	ImageWriteRequest	Write either the image descriptor or image data to a ROM repository. See Table 405 "ImageWriteRequest Message" on page 1392.
0x06	ImageDeleteRequest	Delete an image from the repository. See Table 407 "ImageDeleteRequest Message" on page 1395
0x07 - 0x7F	-	Reserved.
0x80	RepositoryInfoResponse	Response to RepositoryInfoRequest message. See Table 391 "RepositoryInfoResponse Message" on page 1370.
0x81	DescriptorReadResponse	Response to DescriptorReadRequest message. See Table 398 "DescriptorReadResponse Message" on page 1379.
0x82	ImageReadResponse	Response to ImageReadRequest message. See Table 400 "ImageReadResponse Message" on page 1382.
0x83	ImageAddResponse	Response to ImageAddRequest message. SeeTable 402 "ImageAddResponse Message" on page 1386.
0x84	ImageUpdateResponse	Response to ImageUpdateRequest message. See Table 404 "ImageUpdateResponse Message" on page 1391.
0x85	ImageWriteResponse	Response to ImageWriteRequest message. See Table 406 "ImageWriteResponse Message" on page 1392.
0x86	ImageDeleteResponse	Response to ImageDeleteRequest message. See Table 408 "ImageDeleteResponse Message" on page 1395.
0x87 - 0xFF	-	Reserved.

CA5-45: An IOU that provides a ROM repository shall support ROM repository messages specified in Table 389 on page 1368.

A5.12.7 READING ROM REPOSITORY INFORMATION

CA5-46: An IOU that provides a ROM Repository shall respond to RepositoryInfoRequest message as specified in Table 390 on page 1370 with a RepositoryInfoResponse message as specified in Table 391 on page 1370.

The requestor sends a RepositoryInfoRequest message as specified in Table 390 on page 1370 and the IOU responds with a single RepositoryInfoResponse message as specified in Table 391 on page 1370.

Table 390 RepositoryInfoRequest Message

Component	Byte Offset	Length (Bytes)	Description
Opcode	0	1	0x00 - Request information on the ROM repository from the IOU.
Reserved	1	1	Reserved.
RepositoryID	2	2	Identifies the type of ROM repository. See Table 394 on page 1375
TransactionID	4	8	This value is supplied by the requestor and returned in the response.
MaxIdleTime	12	4	Unsigned integer value expressed in 10 millisecond intervals by the requestor to inform an IOU the expected maximum idle time between requests.

Table 391 RepositoryInfoResponse Message

Component	Byte Offset	Length (Bytes)	Description
Opcode	0	1	0x80 - Response to RepositoryInfoRequest message.
Status	1	1	Transaction Status - see Table 392 on page 1372.
RepositoryID	2	2	The value from the request.
TransactionID	4	8	The value from the request.
IdleTimeOut	12	4	Unsigned integer value expressed in 10 millisecond intervals to inform a requestor the idle time the IOU allows between requests before it releases the connection with that requestor.
NumImages	16	4	Total number of images in the repository.
InputBufferSize	20	4	Maximum send payload the IOU can receive in a single send message from a requestor for ROM repository access.
OutputBufferSize	24	4	Maximum send payload the IOU can send in a single send message in response to ROM repository request messages from a requestor.
FreeSpace	28	8	Unused space in the repository.

Table 391 RepositoryInfoResponse Message (Continued)

Component	Byte Offset	Length (Bytes)	Description
Capabilities	32	4	Specifies the capabilities supported by the IOU. Byte:Bit position 0:0 - IOU supports multiple ROM repository connections if this bit is set. 0:1 - IOU supports updates in retain mode if this bit is set. All other bits are reserved and shall be zero.

The repository information messages provides for a mechanism to negotiate idle connection time out value between a requestor and an IOU. The requestor specifies its expected maximum idle time between requests in the MaxIdleTime component of the RepositoryInfoRequest message and the IOU responds with the idle time it allows by specifying the time in the IdleTimeOut component of the RepositoryInfoResponse message.

The idle time allowed by the IOU is not required to be the same as the time out requested by the requestor. A requestor may use the value specified by IdleTimeOut component (see Table 391 on page 1370) to implement some kind of keep-alive mechanism. The action taken by a requestor when an IOU's allowed idle time is less than what it has requested is implementation specific and outside the scope of this specification. How an IOU or a requestor decides on maximum idle time between requests for a given connection is outside the scope of this specification.

The FreeSpace component in Table 391 on page 1370 specifies the unused space available in the repository page identified by the RepositoryID component in Table 390 on page 1370.This is a snap shot of the free space available at the time of response. When the IOU supports multiple connections the available free space can change depending on the operations on other RCs. This component is a useful indicator in single connection implementations.

A requestor accessing a ROM repository can send requests on a connection without waiting for the response to a previous request. The requestor may post receive buffers of varying size depending on the expected response size. For proper receive buffer usage at the requestor's end, an I/O unit is required to send responses to requests in the order they are received.

CA5-47: An IOU that provides a ROM Repository shall send responses to ROM repository requests in the order the requests are received on a given connection.

CA5-48: An IOU that provides a ROM Repository shall support a minimum buffer size of 4096 bytes to receive ROM repository request messages.

CA5-49: An IOU that provides a ROM Repository shall support a minimum buffer size of 4096 bytes to send ROM repository response messages.

Table 392 Transaction Status Codes for All Transactions

Status Code	Description
0x00	**Success** - Indicates that the operation has completed without error.
0x01	**Success With Recovery** - Indicates that the command completed successfully with some recovery action performed. A requestor's action upon receiving this status is implementation specific and outside the scope of this specification.
0x02	**Hardware Error** - Detected a non-recoverable hardware error.
0x03	**Illegal Request** - Illegal parameter in request message or unknown message type.
0x04	**Data Write Protect** - Update prohibited.
0x05	**Data Read Protect** - Read prohibited.
0x06	**Update In Progress** - Indicates that the image is being updated. This status is returned to prevent simultaneous updating of the same image by two requestors and/or to prevent deleting of an image that is being updated.
0x07	**Insufficient Space** - Indicates that there is not enough space in the repository to accommodate the new image.
0x08	**Insufficient Retain space** - Indicates that the IOU does not have enough space to retain the old image while the update is in progress.
0x09	**Invalid Image Handle** - Image handle is invalid. An image handle can become invalid if the corresponding image gets deleted or updated.
0x0A- 0xFF	Reserved.

Data returned with error status i.e status codes other than 0x00 and 0x01 should be discarded.

An IOU returns "Illegal Request" status in the following cases:

1) Operation code specified by the Opcode component in a request is not valid i.e. the specified opcode is not one of the request message opcodes defined inTable 389 on page 1368.
2) The ROM repository type specified by the RepositoryID component in a request is not supported by the IOU.

3) The index number specified by the StartIndex component in a DescriptorReadRequest message (seeTable 397 on page 1379) is outside the valid range. The valid range for a descriptor index number is 0 to n-1, where n is equal to the number of images present in the repository as specified by the NumImages component in RepositoryInfoResponse message (seeTable 391 on page 1370).
4) The size specified by the BufferSize component in a DescriptorReadRequest message (seeTable 397 on page 1379) is equal to zero.
5) The size specified by the BufferSize component in a DescriptorReadRequest message (see Table 397 on page 1379) is greater than the size specified by the OutputBufferSize component in RepositoryInfoResponse message (see Table 391 on page 1370).
6) The NumDescriptors component in a DescriptorReadRequest message (see Table 397 on page 1379) is equal to zero.
7) An ImageReadRequest exceeds the IOU's capacity as specified in the OutputBufferSize component of the RepositoryInfoResponse message (see Table 391 on page 1370). This occurs when the sum of the ByteCount component in the ImageReadRequest message (see Table 399 on page 1381) plus 28 bytes for the ImageReadResponse header (i.e. the byte offset of the Data component from Table 400 on page 1382) is greater than the specified OutputBufferSize.
8) The ImageSize component in an ImageUpdateRequest message (see Table 403 on page 1389) is equal to zero.
9) The DescriptorSize component in an ImageUpdateRequest message (see Table 403 on page 1389) is equal to zero.
10) The Retain component in an ImageUpdateRequest message (see Table 403 on page 1389) specifies a value of 0x01 and the IOU does not support updates in retain mode (see Table 391 on page 1370, Capabilities component).
11) The ImageSize component in an ImageAddRequest message (see Table 401 on page 1385) is equal to zero.
12) The DescriptorSize component in an ImageAddRequest message (see Table 401 on page 1385) is equal to zero.
13) The FinalRequest component in an ImageWriteRequest message (see Table 405 on page 1392) specifies a value other than 0x00 and 0x01.
14) The DataType component in an ImageWriteRequest message (see Table 405 on page 1392) specifies a value other than 0x00 and 0x01.
15) The offset specified by the ByteOffset component in an ImageWriteRequest message (see Table 405 on page 1392) is greater than

n-1, where n is the descriptor size in bytes if the DataType specified is equal to 0x00 or image size in bytes if the DataType specified is equal to 0x01.

16) The sum of the values specified by the ByteOffset component and ByteCount component in an ImageWriteRequest message (see Table 405 on page 1392) is greater than n-1, where n is the descriptor size in bytes if the DataType specified is equal to 0x00 or image size in bytes if the DataType specified is equal to 0x01.

17) The key specified by the AuthenticationKey component in ImageAddRequest (see Table 401 on page 1385) message is an illegal key.

18) The key specified by the AuthenticationKey component in ImageUpdateRequest (see Table 403 on page 1389) message is an illegal key.

19) The key specified by the AuthenticationKey component in ImageDeleteRequest (see Table 407 on page 1395) message is an illegal key.

CA5-50: An IOU that provides a ROM repository shall respond with "Illegal Request" status in all the above mentioned cases.

An IOU is not required to do a boundary check if the operation being performed is an image read or descriptor read operation. An IOU may choose not to check boundary conditions. If a check is done and fails, then the IOU returns a status of 0x03. An IOU's response (in the absence of boundary check) to a read operation beyond an image boundary or descriptor boundary is outside the scope of this specification.

The matrix of possible status codes in response to various request message operation codes (see Table 389 on page 1368) is as shown inTable 393 on page 1374.

Table 393 Status Code Matrix

Status Code / Opcode	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09
0x00	X	X	X	X						
0x01	X	X	X	X						
0x02	X	X	X	X		X				X
0x03	X	X	X	X	X			X		

Table 393 Status Code Matrix (Continued)

Status Code Opcode	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09
0x04	X	X	X	X	X		X	X	X	X
0x05	X	X	X	X	X					X
0x06	X	X	X	X	X					X

Table 394 ROM Repository IDs

Value	Description
0x0000	ROM repository page containing IOC driver images.
0x0001 - 0x7FFF	Reserved for future IBTA definition.
0x8000 - 0xFFFF	**IOU Vendor Specific** - IOU vendor may support IDs in this range to allow access to objects such as micro code or configuration tables in IOUs from that vendor.

A5.12.8 IMAGE DESCRIPTOR

Each image in the Option ROM page has one image descriptor associated with it. Image descriptor contains important information about the associated image such as size of the image, state of the image, image CRC, and information to match the image to a particular hardware and host execution environment. Image descriptor format is as shown in Table 395 on page 1376.

Note that the image descriptor format in Table 395 on page 1376 has been defined primarily for image descriptors in Option ROM page. Other repository types i.e. non Option ROM pages (IOU's vendor specific repository spaces) might not use the same image descriptor format.

Table 395 Image Descriptor Format

Component	Byte Offset	Length (Bytes)	Description
DescriptorSize	0	4	Size of the Image descriptor.
ImageState	4	1	Specifies state of the image as follows: 0x00 - Image is valid. 0x01 - Image is invalid. 0x02 - 0xFF Reserved.
Reserved1	5	3	Reserved.
ImageHandle	8	8	Handle assigned by the IOU to the image associated with this descriptor. A requestor uses this handle to perform operations (read, update or delete) on the image.
ImageSize	16	4	Size of the image.
ImageAuthority	20	4	Specifies the organization that defines the image type and the image content. Byte 0 Bytes 1-3 0x000x000000 IBTA 0x01IEEE OUI All other values - Reserved.
ImageType	24	4	Image type as defined by Image Authority if ImageAuthority component contains an IEEE assigned OUI. If the Image Authority is IBTA, then the ImageType indicates the execution environment as specified by IBTA. See Table 396 on page 1377.
ImageVersion	28	2	Version of the image.
Reserved2	30	2	Reserved.
ImageCRC	32	4	Cyclic Redundancy Code covering all bytes in the image. Calculation is identical to that described for Invariant CRC in the Link Layer Chapter.
PrivateData	36	40	Vendor-defined text description and/or notes (such as build date).
Reserved3	76	2	Reserved.
NumCompStrs	78	2	Number of compatibility strings present in the descriptor. See I/O Annex Section 1.3.1 for information on compatibility strings.
CompStrArray	80	64 * {NumCompStrs}	Array of compatibility strings (for matching driver to hardware).

Each element in the CompStrArray is 64 bytes in size and contains a variable size null terminated compatibility string (see I/O Annex section 1.3.1).

CA5-51: The format of each compatibility string in the CompStrArray component (see Table 395 on page 1376) of an image descriptor present in the Option ROM page shall be as specified in Annex I/O, Table 3 compatibility strings, if the image authority specified by the ImageAuthority component in that image descriptor is IBTA (i.e. a value of 0x00000000).

The ImageAuthority component in Table 395 on page 1376 specifies the organization that defines the image type and the image content. If the ~~least~~ most significant byte (byte 0) is equal to 0x00 then the image authority is IBTA. If it is equal to 0x01 then the image authority is the organization whose IEEE assigned OUI has been specified in bytes 1-3.

The association between the ImageHandle component in Table 395 on page 1376 and its corresponding image does not change during the life time of a given connection. The same association might not exist on a different connection.

Table 396 IBTA Boot Image Types

Value	Description
0x00	Unspecified
0x01	BIOS
0x02	EFI
0x03	IEEE 1275
0x04	PA - IODC
All others	Reserved.

Any organization that has an IEEE assigned OUI may define image types by placing their OUI in the ImageAuthority component. In this case, that organization defines the meaning of the ImageType values.

A5.12.9 READING AN IMAGE DESCRIPTOR

Image descriptors are indexed from 0 to n-1, where n is the total number of image descriptors present in the repository. The total number of image descriptors present in the repository is equal to the value returned in the "NumImages" component of the RepositoryInfoResponse message (see Table 391 on page 1370). Requestor reads image descriptors using index numbers and determines from the image descriptor components, whether or not a desired image is present in the repository.

To read a single image descriptor with a particular index, the requestor specifies the index number in the StartIndex component and specifies a count of one in the NumDescriptors component (see Table 397 on page 1379). To read a range of image descriptors the requestor specifies the

index number of the starting descriptor (of the range) in the StartIndex component and the number of descriptors to be read in the NumDescriptors component.

Since the image descriptor size is variable, it is possible to have an image descriptor that is larger in size than the output buffer size supported by an IOU or the Input buffer size supported by a requestor.To read such an image descriptor, the requestor sends multiple DescriptorReadRequest messages with appropriate offset values specified in the ByteOffset component of each request message. The requestor also specifies the maximum send payload size it can receive in the Buffer Size component of the DescriptorReadRequest message. Since the requestor may not be aware of the descriptor size (particularly when it is reading the descriptor starting from byte0 i.e offset 0), the IOU sends as much descriptor data as it can send without exceeding the size specified in the BufferSize component of the request.

To prevent discontinuity in the descriptor index range, an IOU does not delete image descriptors corresponding to deleted or updated images as long as there is a connection to the ROM repository. The IOU simply changes the ImageState component in the descriptor to “invalid” state.

CA5-52: An IOU that provides a ROM repository shall not respond with a DescriptorReadResponse message that is larger in size than the value specified by the BufferSize component in the DescriptorReadRequest message.

CA5-53: An IOU that provides a ROM repository shall not change the index number assignments of existing image descriptors (as viewed by the requestor) as long as a given RC exists.

CA5-54: An IOU that provides a ROM Repository shall respond to DescriptorReadRequest message as specified in Table 397 on page 1379 with a DescriptorReadResponse message as specified in Table 398 on page 1379.

The requestor sends DescriptorReadRequest message as specified in Table 397 on page 1379 and the IOU responds with DescriptorReadResponse message as specified in Table 398 on page 1379. The ladder diagram of image descriptor read operation is as shown in Figure 282.

Table 397 DescriptorReadRequest message

Component	Byte Offset	Length (Bytes)	Description
Opcode	0	1	0x01- Request one or more image descriptors from the IOU's ROM repository.
Reserved1	1	1	Reserved.
RepositoryID	2	2	Identifies the type of ROM repository. See Table 394 on page 1375.
TransactionID	4	8	This value is supplied by the requestor and returned in the response.
StartIndex	12	4	Index of the image descriptor to read or index of the first image descriptor if reading multiple image descriptors.
ByteOffset	16	4	Specifies the byte offset from the start of the image descriptor where the transfer starts. The specified offset is only for the image descriptor referenced by the StartIndex component. Note that descriptor Byte 0 is at Offset 0x00.
NumDescriptors	20	4	Number of image descriptors to read.The starting image descriptor is specified in the StartIndex component.
BufferSize	24	4	Specifies the buffer size allocated by the requestor to receive the response to this request message.

Table 398 DescriptorReadResponse Message

Component	Byte Offset	Length (Bytes)	Description
MessageType	0	1	0x81- Response to DescriptorReadRequest message.
Status	1	1	Transaction Status. See Table 392 on page 1372.
RepositoryID	2	2	The value from the request.
TransactionID	4	8	The value from the request.
StartIndex	12	4	The value from the request.
ByteOffset	16	4	The value from the request.
NumDescriptors	20	4	Actual number of image descriptors sent in this message. This number includes partial descriptors if any.
Reserved1	24	4	Reserved.
ByteCount	28	4	Number of bytes sent in the Data component.

Table 398 DescriptorReadResponse Message (Continued)

Component	Byte Offset	Length (Bytes)	Description
Data	32	{ByteCount}	This component contains full or partial image descriptor data. It contains tightly packed image descriptors if the IOU returns more than one descriptor.

The NumDescriptors component in the DescriptorReadResponse message (see Table 398 on page 1379) specifies actual number of image descriptors returned in the Data component. This number can be less than the number specified in the corresponding request message (see Table 397 on page 1379), when the requested number of descriptors can't fit in a single response message and/or when there are not as many image descriptors (starting with the StartIndex, see Table 397 on page 1379) as requested by the requestor.

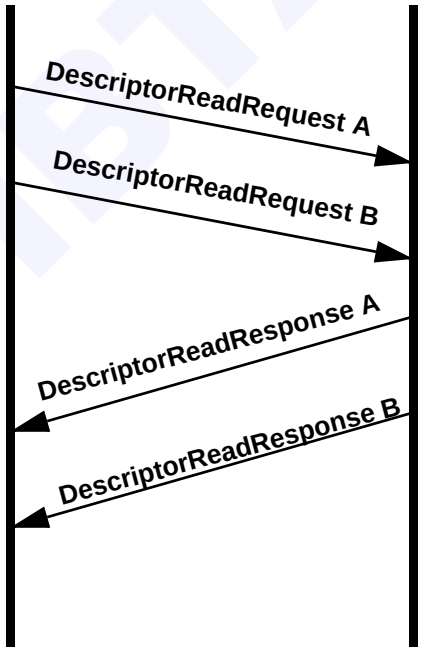


Figure 282 Ladder Diagram of Image Descriptor Read Operation

A5.12.10 READING AN IMAGE

A requestor selects a particular image for reading by matching one of the IOC's compatibility strings with one of the I/O driver's compatibility strings (see CompStrArray component in Table 395 on page 1376), for image de-

scriptors that have an ImageAuthority, ImageType, and ImageVersion matching the boot environment. The requestor then uses the image handle present in the matching descriptor to read the image.The requestor sends an ImageReadRequest message as specified in Table 399 on page 1381 and the IOU responds with a single ImageReadResponse message as specified in Table 400 on page 1382. The ladder diagram of an image read operation is as shown in Figure 283.

An image and its image handle are valid only when the ImageState component in the corresponding image descriptor is equal to 0x00.

CA5-55: An IOU that provides a ROM Repository shall respond to an ImageReadRequest message as specified in Table 399 on page 1381 with an ImageReadResponse message as specified in Table 400 on page 1382.

CA5-56: An IOU that provides a ROM Repository shall respond with the status “Invalid Image Handle”, if the image handle specified in a message sent by the requestor is not a valid image handle.

An image handle becomes invalid, if the image gets deleted or updated. When there are two or more requestors accessing the ROM Repository, a requestor can initiate image update or image delete operation even if that image is currently being read by other requestors. An image is invalidated by writing a value of 0x01 in the ImageState component of the corresponding image descriptor.

Table 399 ImageReadRequest Message

Component	Byte Offset	Length (Bytes)	Description
Opcode	0	1	0x02 - Request image data from the IOU.
Reserved1	1	1	Reserved.
RepositoryID	2	2	Identifies the type of ROM repository. See Table 394 on page 1375
TransactionID	4	8	This value is supplied by the requestor and returned in the response.
ImageHandle	12	8	Image handle specified in the image descriptor.
ByteOffset	20	4	Specifies the byte offset from the start of the image where the transfer starts. Byte 0 is at Offset 0x00.
ByteCount	24	4	Specifies the number of bytes of image data to transfer.

Table 400 ImageReadResponse Message

Component	Byte Offset	Length (Bytes)	Description
Opcode	0	1	0x82 - Response to ImageReadRequest message.
Status	1	1	Transaction Status. See Table 392 on page 1372.
RepositoryID	2	2	Value from the request.
TransactionID	4	8	Value from the request.
ImageHandle	12	8	Value from the request.
ByteOffset	20	4	Value from the request.
ByteCount	24	4	Actual number of bytes returned in the data component.
Data	28	{ByteCount}	Image data.

The requestor should check for under-flows by comparing the returned byte count (ByteCount component in the ImageReadResponse message) with the byte count requested in ImageReadRequest message.

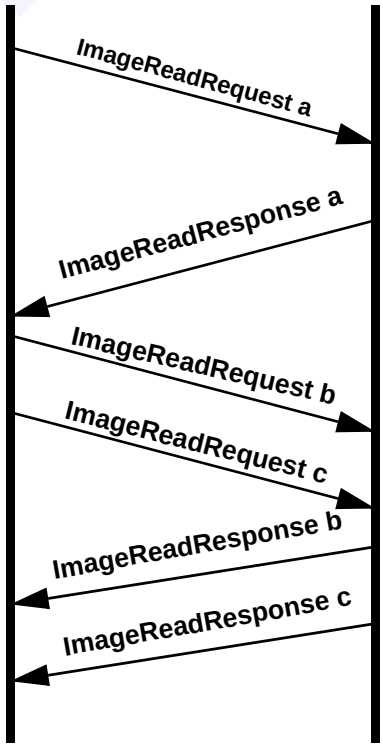


Figure 283 Ladder Diagram of Image Read Operation

A5.12.11 ADDING AND UPDATING AN IMAGE

A requestor can modify a ROM repository by performing operations such as adding new images to the repository, modifying (updating) images present in the repository, and deleting images from the repository.

There are two possible modes to update an image in the repository. One is to invalidate the existing image (make it unavailable to other requestors) before writing the new image and the other mechanism is to retain the existing image (and make it available to other requestors) until the modified image is written successfully. The location at which the new image is stored in the repository is implementation specific.

The Retain component in the ImageUpdateRequest (seeTable 403 on page 1389) message specifies which mode to use. Update support in retain mode is optional and the Capabilities component in the RepositoryInfoResponse message (see Table 403 on page 1389) specifies whether or not the IOU supports updates in retain mode.

The ROM repository protocol provides an authentication mechanism for I/O units to authenticate image add, image update, and image delete operations by checking the Authentications component specified in ImageAddRequest (see Table 401 on page 1385), ImageUpdateRequest (see Table 403 on page 1389), and ImageDeleteRequest (see Table 407 on page 1395) messages. How a requestor obtains the authentication key is outside the scope of this annex. An IOU is not required to authenticate image add, image update and image delete operations. An IOU may accept any authentication key in the request message, if it does not support authentication of the above mentioned operations.

A ROM repository supported by an IOU can be write protected. How write protection is enabled or disabled is outside the scope of this annex. If a repository is write protected, the IOU responds to ImageAddRequest, ImageUpdateRequest, ImageWriteRequest, and ImageDeleteRequest, messages with “Data Write Protected” (seeTable 392 on page 1372) status.

A5.12.11.1 INITIATING AN IMAGE ADD OPERATION

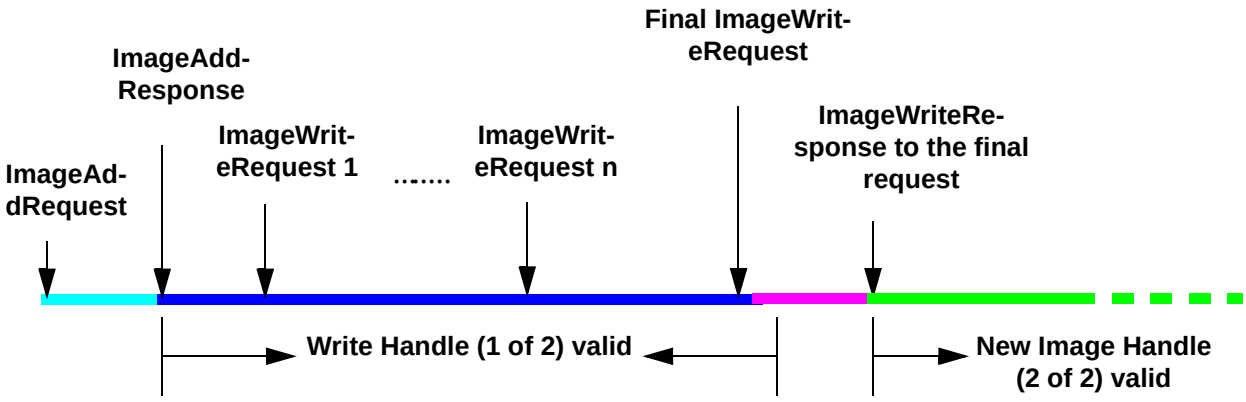
A requestor initiates an image add operation by sending ImageAddRequest message to the IOU as specified in Table 401 on page 1385. The requestor specifies image size and image descriptor size in the request message. The IOU responds with ImageAddResponse message as specified in Table 402 on page 1386.The response message contains the write handle allocated by the IOU to add the new image and its image descriptor. This write handle is unique across all connections. The requestor uses the allocated write handle to write the new image and new image descriptor as specified in A5.12.11.3 "Writing Descriptor and Image Data" on page 1391.

Following are the steps involved in adding an image:

1) The requestor sends ImageAddRequest message (seeTable 401 on page 1385) specifying the authentication key, image descriptor size, and image size.
2) The IOU allocates a Write Handle[36] and returns it in the response message (see Table 402 on page 1386).
3) The requestor sends ImageWriteRequest messages (see Table 405 on page 1392) specifying the Write Handle (returned in step 2), Byte Offset, Byte Count, etc. to write the new image descriptor and new image.
4) The requestor informs an IOU that it has sent all the bytes of image descriptor and image by specifying a value of 0x01 in the Final Request component (see Table 405 on page 1392).
5) When the IOU receives an ImageWriteRequest message with Final Request component equal to 0x01, it completes the writing of new image and descriptor. If the write is successful, then the IOU allocates a new handle, updates the image descriptor with the new handle[37], and then makes the new image descriptor available for reading.

There are two handles involved in an image add operation. The scope of these handles during an add operation is as shown in Figure 284.

36. The IOU assigns a unique handle for adding the image.
37. The IOU assigns another unique handle for all future access.



- Initiation of image add operation
- Writing of image descriptor and image data
- IOU allocates new image handle (2 of 2), invalidates Write handle (1 of 2) and completes writing of image descriptor and image data.
- Image add operation is successful. New image handle (2 of 2) and the new image are available to the requestors.

Figure 284 Handles Involved in an Image Add Operation

CA5-57: An IOU that provides a ROM Repository shall respond to an ImageAddRequest message as specified in Table 401 on page 1385 with an ImageAddResponse message as specified in Table 402 on page 1386.

CA5-58: An IOU that provides a ROM repository shall respond with the status “Insufficient Space” (see Table 392 on page 1372) to an ImageAddRequest message, if there is not enough space to store the new image and the new descriptor.

Table 401 ImageAddRequest Message

Component	Byte Offset	Length (Bytes)	Description
Opcode	0	1	0x03 - Initiate an image add operation.
Reserved1	1	1	Reserved.
RepositoryID	2	2	Identifies the type of ROM repository. See Table 394.
TransactionID	4	8	This value is supplied by the requestor and returned in the response.

Table 401 ImageAddRequest Message (Continued)

Component	Byte Offset	Length (Bytes)	Description
Reserved2	12	8	Reserved.
ImageSize	20	4	Size of the image to be added.
DescriptorSize	24	4	Size of the image descriptor.
AuthenticationKey	28	8	Key for the IOU to authenticate the add operation.

Table 402 ImageAddResponse Message

Component	Byte Offset	Length (Bytes)	Description
Opcode	0	1	0x83 - Response to ImageAddRequest message.
Status	1	1	Transaction Status See Table 392 on page 1372.
RepositoryID	2	2	Value from the request
TransactionID	4	8	Value from the request.
WriteHandle	12	8	Image handle allocated by the IOU to add the new image. This handle is used by the requestor to write the new image and new descriptor.

A5.12.11.2 INITIATING AN IMAGE UPDATE

A requestor initiates an image update operation by sending ImageUpdateRequest message to the IOU as specified in Table 403 on page 1389. The IOU responds with ImageUpdateResponse message as specified inTable 404 on page 1391.The response message contains the image handle allocated by the IOU to update the existing image and its image descriptor. This image handle is unique across all connections. The requestor uses the allocated image handle to write the new image and new image descriptor as specified in A5.12.11.3 "Writing Descriptor and Image Data" on page 1391.

Following are the steps involved in updating an image:

1) The requestor sends ImageUpdateRequest (see Table 403 on page 1389) message specifying the image handle[38] of the image being modified, new image size, new descriptor size, authentication key, etc.

38. Image handle of the existing image from the corresponding descriptor.

2) The IOU allocates Write Handle[39] (if the update is allowed and there are enough resources) and returns it in the response message (see Table 404 on page 1391).The IOU invalidates the old image before sending the response (with success status), if the requestor specifies a value of 0x00 in the Retain component of the corresponding request (see Table 403 on page 1389).

3) The requestor then sends ImageWriteRequest messages (see Table 405 on page 1392) specifying the Write Handle (returned in step 2), Byte Offset, Byte Count etc. to write the modified image descriptor and modified image.

4) The requestor informs an IOU that it has sent all the bytes of image descriptor and image by specifying a value of 0x01 in the Final Request component (see Table 405 on page 1392).

5) When the IOU receives an ImageWriteRequest message with Final-Request component equal to 0x01, it completes the writing of modified image and descriptor. If the write is successful then the IOU allocates a new handle[40] and updates the image descriptor with the new handle.

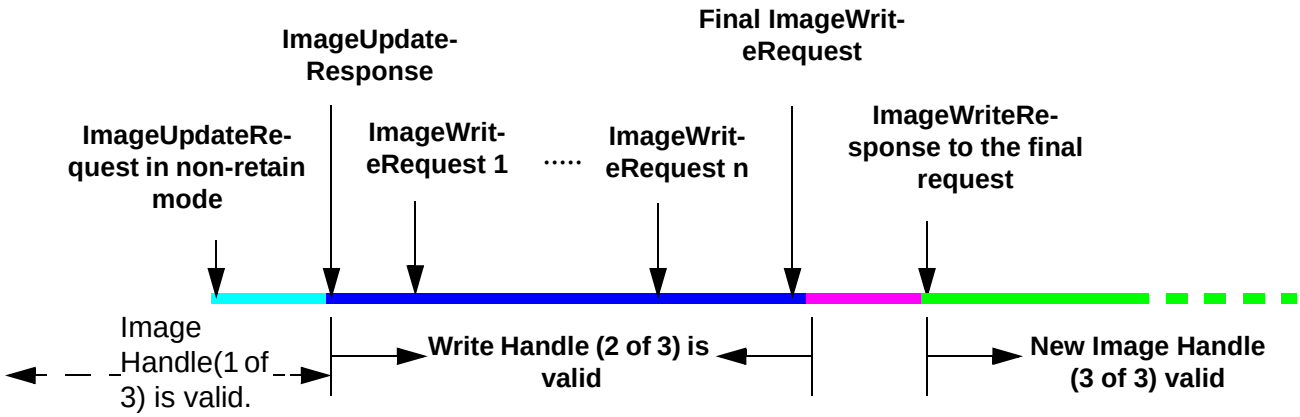
6) If the update is initiated in the Retain mode (see Retain component in Table 12) then the IOU invalidates the old image by assigning a value of 0x01 to the ImageState component in the old image descriptor. The IOU does not delete or over-write the old image descriptor as long as there is a connection to the ROM repository.

7) The IOU makes the new image descriptor and new image available for reading.

Updating an image always requires writing the image descriptor as well as the image. ImageWriteRequest message is used for writing the image and also the image descriptor. The DataType component in the Image-WriteRequest message (see Table 405 on page 1392) specifies whether the Data component contains image data or descriptor data.

There are three handles involved in an image update operation. The scope of these handles during an update operation in non retain mode is as shown in Figure 285.

39. IOU assigns a unique handle to update the image.
40. IOU assigns another unique handle for all future access.



Requestor initiates image update operation with the image handle (1 of 3) of the image to be modified. IOU allocates Write Handle (2 of 3) and invalidates the old handle (1 of 3).

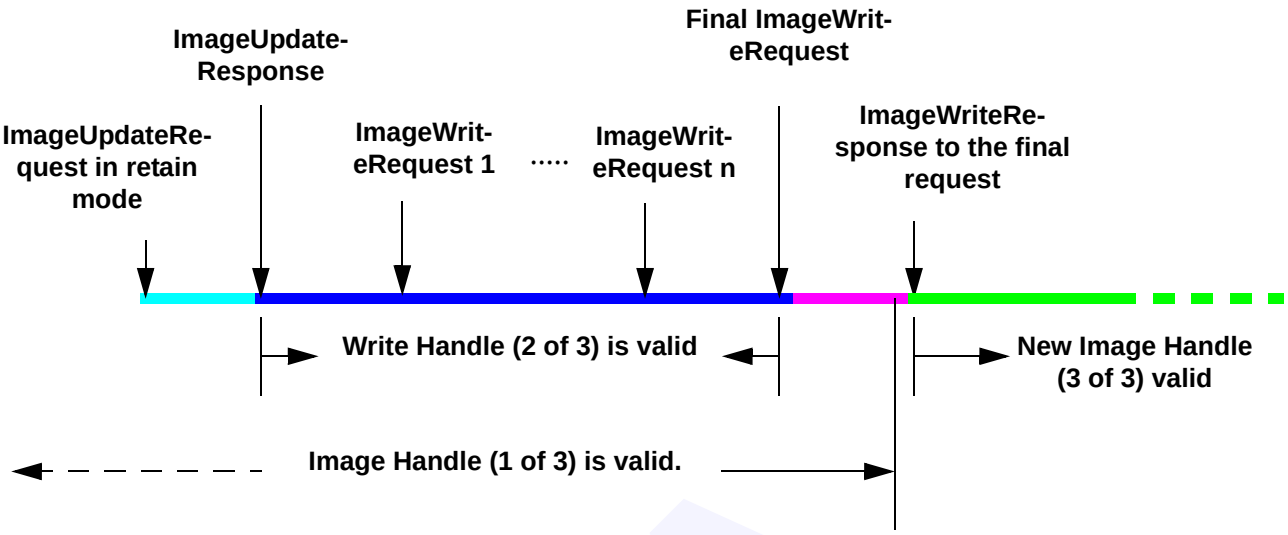
Writing of image descriptor and image data

IOU invalidates Write handle (2 of 3), allocates new image handle (3 of 3) and completes writing of image descriptor and image data.

Image add operation is successful. New image handle (3 of 3) and the updated image are available to the requestors.

Figure 285 Handles Involved in an Image Update in Non Retain Mode

The scope of the handles during an update operation in retain mode is as shown in Figure 286.



- Requestor initiates image update operation with the image handle (1 of 3) of the image to be modified. IOU allocates Write Handle (2 of 3).
- Writing of image descriptor and image data
- IOU invalidates Write handle (2 of 3), allocates new image handle (3 of 3), completes writing of image descriptor and image data and invalidates image handle of the old image (1 of 3).
- Image add operation is successful. New image handle (3 of 3) and the updated image are available to the requestors.

Figure 286 Handles Involved in an Image Update in Retain Mode

Table 403 ImageUpdateRequest Message

Component	Byte Offset	Length (Bytes)	Description
Opcode	0	1	0x04 - Initiate an image update operation.
Reserved1	1	1	Reserved.
RepositoryID	2	2	Identifies the type of ROM repository. See Table 394 on page 1375
TransactionID	4	8	This value is supplied by the requestor and returned in the response.
ImageHandle	12	8	Image handle of the image being updated as specified by the Image-Handle component in the corresponding image descriptor.

Table 403 ImageUpdateRequest Message (Continued)

Component	Byte Offset	Length (Bytes)	Description
Reserved2	20	8	Reserved.
Retain	28	1	Specifies the update mode. Value 0x00 - Update in non-retain mode. Value 0x01 - Update in retain mode Values 0x02 - 0xFF - Reserved.
Reserved3	29	3	Reserved.
ImageSize	32	4	Size of the new image.
DescriptorSize	36	4	Size of the new image descriptor.
AuthenticationKey	40	8	Key for the IOU to authenticate the update operation.

The Retain component in Table 403 on page 1389 specifies whether or not the IOU should retain the old image (descriptor, handle and image data for reading by other requestors) until the update process completes successfully (good response to the final ImageWriteRequest message). A value 0x00 in the Retain component specifies that the IOU should invalidate the old image immediately, if the requestor is allowed to update the old image. This is a destructive upgrade because the old image is lost even if the upgrade fails for any reason. A value of 0x01 specifies that the IOU should not invalidate the old image until the update completes successfully. In this case, if the update fails for any reason, the old image is still available and valid.

Note that the IOU lists the new image descriptor only when the update is successful in either case.

CA5-59: An IOU that provides a ROM Repository shall respond to an ImageUpdateRequest message as specified in Table 403 on page 1389 with an ImageUpdateResponse message as specified in Table 404 on page 1391.

CA5-60: An IOU that provides a ROM Repository shall respond with the status “Update In Progress” to an ImageUpdateRequest message, if the image is currently being updated.

CA5-61: An IOU that provides a ROM repository shall respond with the status “Insufficient Space” (see Table 392 on page 1372) to an ImageUpdateRequest message, if there is not enough space to store the new image and the new descriptor because of the increase in size.

CA5-62: An IOU that provides a ROM Repository shall respond with the status “Insufficient Retain Space” to an ImageUpdateRequest message with Retain component equal to 0x01, if there is not enough space to retain the old image, but there is enough space to store the new image and new descriptor without retaining the old image.

CA5-63: An IOU that provides a ROM repository shall invalidate the old image before responding with a Success status to an ImageUpdateRequest message with Retain component equal to 0x00.

Table 404 ImageUpdateResponse Message

Component	Byte Offset	Length (Bytes)	Description
Opcode	0	1	0x84 - Response to ImageUpdateRequest message.
Status	1	1	Transaction Status. See Table 392 on page 1372.
RepositoryID	2	2	Value from the request.
TransactionID	4	8	Value from the request.
ImageHandle	12	8	Value from the request.
WriteHandle	20	8	Image handle allocated by the IOU for the update. This handle is used to write the descriptor and image data.

A5.12.11.3 WRITING DESCRIPTOR AND IMAGE DATA

To write a new image or to update an existing image the requestor sends ImageWriteRequest message as specified in Table 405 on page 1392 and the IOU responds with ImageWriteResponse message as specified inTable 406 on page 1392.

The IOU uses the Write Handle to correlate ImageWriteRequest messages with the corresponding add or update operation.

CA5-64: An IOU that provides a ROM repository shall respond to an ImageWriteRequest message as specified in Table 405 on page 1392 with an ImageWriteResponse message as specified in Table 406 on page 1392.

Table 405 ImageWriteRequest Message

Component	Byte Offset	Length (Bytes)	Description
Opcode	0	1	0x05 - Write image and image descriptor data to the ROM repository.
Reserved1	1	1	Reserved.
RepositoryID	2	2	Identifies the type of ROM repository. See Table 394 on page 1375
TransactionID	4	8	This value is supplied by the requestor and returned in the response.
WriteHandle	12	8	Handle allocated for writing the image. This handle is returned in the ImageUpdateResponse message if the operation is an update or returned in the ImageAddResponse message if the operation is an add operation.
FinalRequest	20	1	Specifies whether or not this is the final request message for the image descriptor and image write operation that is currently in progress. Value 0x00 - This message is not the final request. Value 0x01 - This message is the final request. Values 0x02 - 0xFF are reserved.
DataType	21	1	Specifies whether the data contained in the Data component is descriptor data or image data. Value 0x00 - Descriptor data. Value 0x01 - Image data. Values 0x02 - 0xFF - Reserved.
Reserved2	22	2	Reserved.
ByteOffset	24	4	Specifies the byte offset from the start of the image where the transfer starts. Byte 0 is at Offset 0x00.
ByteCount	28	4	Specifies the number of bytes of image data to transfer.
Data	32	{ByteCount}	Image descriptor data or image data for writing.

Table 406 ImageWriteResponse Message

Component	Byte Offset	Length (Bytes)	Description
Opcode	0	1	0x85 - Response to ImageWriteRequest message.
Status	1	1	Transaction Status. See Table 392 on page 1372.
RepositoryID	2	2	Value from the request.

Table 406 ImageWriteResponse Message (Continued)

Component	Byte Offset	Length (Bytes)	Description
TransactionID	4	8	Value from the request.
WriteHandle	12	8	Value from the request.

CA5-65: An IOU that provides a ROM Repository shall allocate a new image handle and shall update the ImageHandle component in the image descriptor before making the new or modified image available to requestors.

CA5-66: An IOU that provides a ROM repository shall not invalidate the old image until the image update is successful, if the corresponding ImageUpdateRequest is sent with a value of 0x01 in the Retain component.

The ladder diagrams of an image update and add operations are as shown in Figure 287.

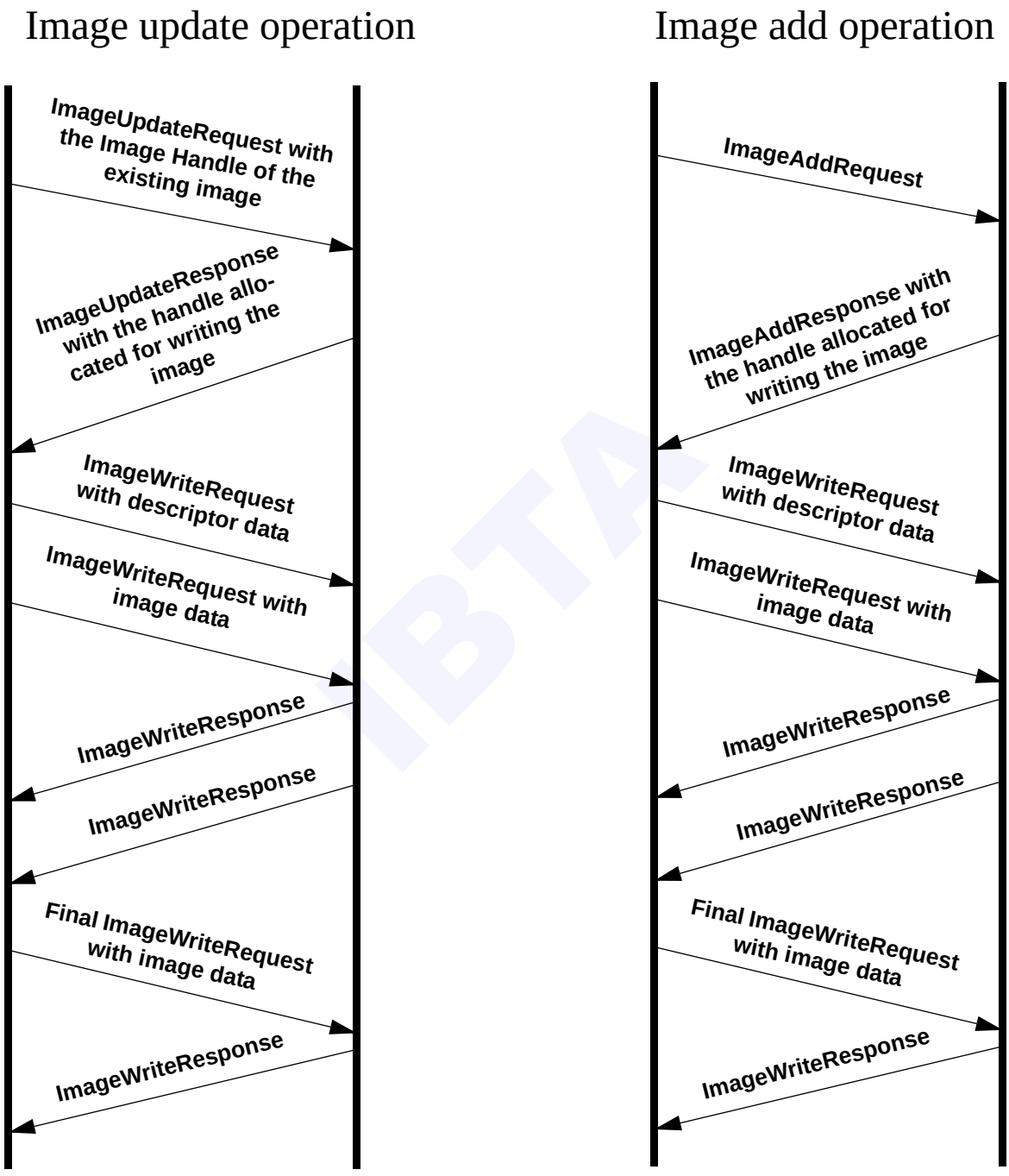


Figure 287 Ladder Diagram of Image Write operation

A5.12.11.4 DELETING AN IMAGE

CA5-67: An IOU that provides a ROM Repository shall respond to an ImageDeleteRequest message as specified in Table 407 on page 1395 with an ImageDeleteResponse message as specified in Table 408 on page 1395.

Table 407 ImageDeleteRequest Message

Component	Byte Offset	Length (Bytes)	Description
Opcode	0	1	0x06 - Delete an image from the repository.
Reserved	1	1	Reserved.
RepositoryID	2	2	Identifies the type of ROM repository. See Table 394 on page 1375
TransactionID	4	8	This value is supplied by the requestor and returned in the response.
ImageHandle	12	8	Image handle of the image to be deleted.
AuthenticationKey	20	8	Key for the IOU to authenticate the delete operation.

Table 408 ImageDeleteResponse Message

Component	Byte Offset	Length (Bytes)	Description
Opcode	0	1	0x86 - Response to ImageDeleteRequest message.
Status	1	1	Transaction Status. See Table 392 on page 1372.
RepositoryID	2	2	Value from the request.
TransactionID	4	8	Value from the request.
ImageHandle	12	8	Value from the request.

CA5-68: An IOU that provides a ROM repository shall respond with the status “Update in Progress” to an ImageDeleteRequest, if the image is currently being updated.

CA5-69: An IOU that provides a ROM repository shall set the ImageState component to a value of 0x01 in the image descriptor corresponding to the deleted image, if and only if, the delete operation is successful.

The IOU immediately frees up the space occupied by an image, if the image delete operation on that image is successful. The IOU does not de-

lete the corresponding image descriptor as long as there is a connection to the ROM repository that is aware of that image descriptor. A connection is assumed to be aware of an image descriptor, if the index number associated with that image descriptor is less than the value returned in the NumImages component of a RepositoryInfoResponse message (see Table 391 on page 1370).

CA5-70: An IOU that provides a ROM repository shall not delete the image descriptor corresponding to deleted/invalidated image as long as there is a connection to the ROM repository that is aware of the image descriptor corresponding to the deleted/invalidated image.

A5.13 COMPLIANCE SUMMARY

A5.13.1 BOOTING SPECIFICATION COMPLIANCE CATEGORIES

In order to claim compliance to the Booting Specification a product shall meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support.

Table 409 Booting Compliance Categories/Qualifiers

Category	Qualifiers	Description
BtA	none	Minimum requirements for a BootAgent
	Trap	Asynchronous Event Notification
	NQ	Notice Queue
	Info	Capable of saving PlatformBootInfo and PortBootInfo attributes into non-volatile storage
	RomLoc	Capable of saving and deleting RomRepositoryLocatorRecords into non-volatile storage
	ConLoc	Capable of saving and deleting ConsoleLocatorRecords into non-volatile storage
	OsLoc	Capable of saving and deleting OsLocatorRecords into non-volatile storage
	Reboot	Capable of rebooting the platform
BtM	none	Minimum requirements for a BootManager
	Subscribe	BootManager supports 3rd party Trap subscriptions
	PERS	BootManager retains subscription information across power cycles and resets - requires compliance with Subscribe qualifier
	FAILOVER	BootManager supports graceful failover to another boot manager - requires compliance with Subscribe qualifier

Table 409 Booting Compliance Categories/Qualifiers (Continued)

Category	Qualifiers	Description
BtPlatform	none	Minimum requirements for a Boot Platform
	Info	Capable of reading PlatformBootInfo and PortBootInfo from non-volatile storage
	NV	Capable of reading RomRepositoryLocatorRecords, ConsoleLocatorRecords and OsLocatorRecords from non-volatile storage
	ExBE	Capable of to expanding the Boot Environment using a ROM Repository
	BIS	Capable of using the BIS service and using BIS class attributed for booting
	NWBoot	Capable of booting using the IB Network model
	BootOS	Capable of booting the OS over the IB Network
	Con	Capable of using the IB attached Console
RomRepos	none	Minimum requirements for ROM Repository

A5.13.2 BOOTAGENT (BTA) COMPLIANCE CATEGORY

In order to claim compliance to the InfiniBand Specification for the Compliance Category of BootAgent (BtA), a product shall meet all requirements specified in this section, in Section 20.12, "Optional Management Agent Compliance Category," on page 1116, and Section 20.14, "Common MAD Requirements," on page 1119, except for those statements preceded by Qualifiers that the product does not support.

A5.13.3 BOOTMANAGER (BTM) COMPLIANCE CATEGORY

In order to claim compliance to the InfiniBand Specification for the Compliance Category of BootManager (BtM), a product shall meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support.

In addition, a Boot Manager must also be compliant with the Common MAD requirements specified in Section 20.14, “Common MAD Requirements,” on page 1119 and the following general management framework requirements from Chapter 13.

A5.13.4 BOOT PLATFORM (BTPLATFORM) COMPLIANCE CATEGORY

In order to claim compliance to the InfiniBand Specification for the Compliance Category of Boot Platform (BtPlatform), a product shall meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support.

A5.13.5 ROM REPOSITORY COMPLIANCE CATEGORY

In order to claim compliance to the InfiniBand Specification for the Compliance Category of ROM Repository (RomRepos), a product shall meet all requirements specified in this section, in Section 20.12, "Optional Management Agent Compliance Category," on page 1116, and Section 20.14, "Common MAD Requirements," on page 1119. This category has no optional qualifiers.

ANNEX A6: BOOT INFORMATION SERVICE

A6.1 INTRODUCTION

This annex is a supplement to Volume 1 of the InfiniBand Architecture Specification, herein referred to as the base document. This annex specifies a Boot Information Service. In particular, it describes how a booting platform requests and receives boot information by defining methods, attributes, and requirements on how the Boot information Service responds.

A6.1.1 GLOSSARY

The following are additional terms not found in the Volume 1 Glossary (Chapter 2).

BIS — Boot Information Service

Boot Information Record — Information that provides booting parameters or describes a device or service that a Booting Platform can use to boot its operating environment.

Boot Information Service — A management agent of a server that supplies boot information records.

I/O Partition — Any IB fabric partition that allows an I/O client (e.g., a host) to communicate with an I/O Unit for the purpose of performing I/O operations.

Booting Platform — An endnode that is booting its operating system using the IB fabric.

A6.1.2 COMPLIANCE

This annex specifies a new Compliance Category (see Chapter 20: Volume 1 Compliance Summary on page 1072 for explanation of compliance categories and qualifiers). The new category is BIS-Server.

There are no Compliance Qualifiers for BIS-Server.

Section A6.5 “Compliance Summary” on page 1434 provides a summary of compliance statements.

A6.2 BIS OVERVIEW

This annex defines an optional management service called Boot Information Service (BIS). A BIS is an IB management agent (i.e., GSA) that provides boot information to booting platforms, which, in the role of a General

Service Manager (GSM), queries the BIS for boot information. Boot information includes booting parameters as well as locator records identifying ROM Repositories, boot devices, and console services.

In general, this annex defines the external behavior of the BIS, including its relationship with other management entities. In particular, it defines the methods and attributes that a booting platform uses to query the BIS and thus retrieve its boot information. The internal behavior of the BIS is outside the scope of this annex.

The Booting Annex specifies a boot management class that permits a boot manager to configure a booting platform with its booting information with the expectation that the platform persistently stores its boot information. As illustrated in Figure 288,the BIS is the complement to that capability where the booting platform may query the BIS each time that it boots. Thus, the booting platform retrieves the latest set of boot information available from the BIS, reducing the need for the boot platform to persistently store boot information. The Booting Annex specifies how a booting platform can support both BIS and persistent storage of boot information.

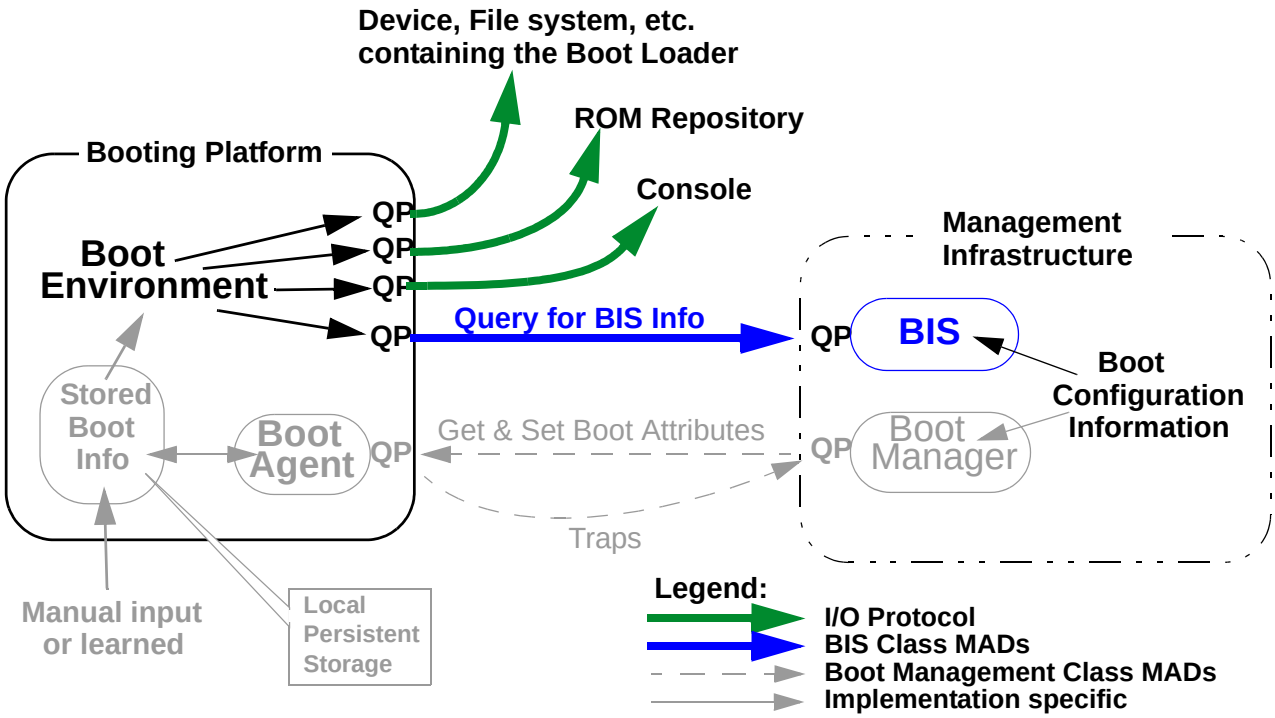


Figure 288 Booting Framework

Whether a booting platform always uses BIS or only uses a BIS if its persistent information fails is a local policy that depends on many factors including the capability of the BIS. The BIS definition permits a broad range of innovation ranging from a BIS that simply stores information and provides it unmodified to the booting platform, to an intelligent implementa-

tion that responds differently based on device availability, subnet loading, fabric failures, etc.

A6.2.1 BIS OPERATIONAL MODEL

A BIS provides boot information to booting platforms at their request. Typically the booting platform is a host trying to bootstrap (load) its operating system, however, nothing prevents an I/O unit from using a BIS in booting its operating environment.

The booting platform must first locate the BIS. This is accomplished by having the BIS register with the SA via SubnAdmSet(ServiceRecord) so that each booting platform can query its SA to find the BIS. Once the booting platform has resolved a path to the BIS, it can query the BIS for its boot information.

Figure 289 illustrates the relationship of a BIS to a booting platform. A booting platform queries the BIS, which provides the booting platform with boot information records describing devices or services needed for booting. As illustrated, the booting platform need not query the BIS through the same port that it uses to access boot devices. The BIS can provide locator records describing devices accessible through any port of the booting platform.

The BIS only provides boot information and does not provide or broker any boot services, such as providing a boot image to the booting platform (i.e., the booting platform does not boot through the BIS). The booting platform uses the BIS information to locate and then establish communications directly with the devices and services as necessary (as illustrated by the dotted lines in Figure 289).

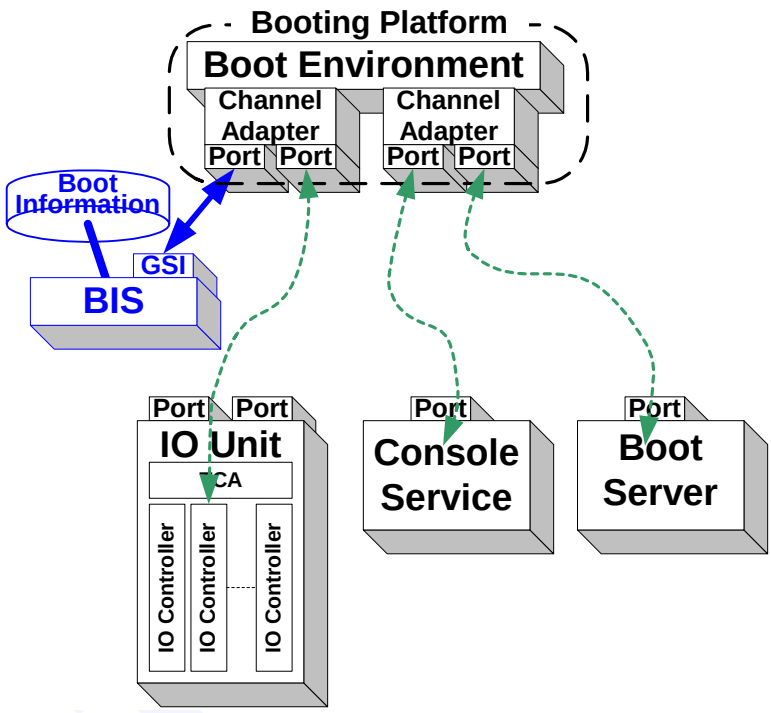


Figure 289 BIS Application Model

When a booting platform queries the BIS, it provides the BIS with:

- A unique identifier (BootPlatformUUID), that uniquely identifies the booting platform.
- Identification string (PlatformInfo) containing vital information that is useful for identifying a new platform.
- Port identifier (PortGUID) that the BIS uses when the booting platform requests port parameters.
- An indication of the type of boot information that the booting platform wants, such as console, storage boot device, ROM Repository for a device driver, etc. One type per request.
- An indication of the booting capabilities that the booting platform supports.

The BIS responds with zero or more boot information records of the type that the booting platform requested.

This specification only specifies the interface between a booting platform and the BIS. Thus, how a BIS determines the boot information is outside the scope of this annex. It is assumed that the BIS has some form of user interface that permits a system administrator to configure the BIS with boot information. In addition, the BIS might be intelligent enough to deal with booting platforms for which it does not have specific information.

A6.2.2 RELATIONSHIP WITH OTHER MANAGEMENT CLASSES

Naturally, there is a strong dependency on the BIS information to be consistent with boot management, I/O configuration, and partition management. How a BIS implementation obtains its data and maintains its integrity to provide data consistent with other management classes is outside the scope of this annex and thus left to the innovation of the BIS vendor. A number of possibilities exists ranging from manually configuring the BIS to integrating the BIS with the other management services.

Architecturally, there are several relationships with other management classes as follows.

A6.2.2.1 BOOT MANAGEMENT

Boot Management (see Booting Annex) and Boot Information Service classes complement each other. Boot management provides the means to manage the booting platform and set parameters that the booting platform needs to locate a BIS. In fact, the booting platform can independently access a BIS for various purposes (finding a console, finding a ROM Repository, finding the OS loader) and the Boot Manager can configure the booting platform for each of these phases. For example, a booting platform can be configured to only use local persistent information to find a ROM Repository, always use the BIS to find its console, and use the BIS to find its the boot loader only if its local persistent information is not valid.

Since a boot manager using Boot Management class MADs is capable of providing the same information as a BIS using BIS class MADs, the attributes for both classes are aligned as follows:

- Attribute components that use the same names have the same definition.
- There is a one to one correlation between attributes used in BisQueryResp() and BootMgtSet() attributes. Note that some BootMgt attributes, such as NodeReboot are not relevant for BIS and thus have no equivalent.
- Attribute formats are consistent in that components in BisQueryResp() attributes have the same offset as their equivalent R/W component in the Boot Management attribute. Thus, code needed to process a BisQueryResp() will be similar to that needed for processing the BootMgtSet() for the equivalent attribute.

A6.2.2.2 SUBNET ADMINISTRATION

BIS uses the facilities of the Subnet Administration class to allow booting platforms to locate a BIS. The BIS is a service that registers with the SA. This registration provides the means for booting platforms to locate the BIS. The SA only allows an authorized BIS to register. This is enforced by the ServiceKey used in the service registration. How a BIS gets authorized (i.e., learns its ServiceKey) is outside the scope of this annex.

A6.2.2.3 SUBNET MANAGEMENT

For a booting platform to use a BIS, that BIS has to share a partition with the booting platform. There is no special P_Key for a BIS. The SM assigns P_Keys to a node containing a BIS the same as it assigns P_Keys to any other node. That is, normal partitioning rules apply and how partitions are assigned is outside the scope of this document.

Additionally, BIS information needs to conform with partitions established by the SM. Again, this is an implementation detail outside the scope of this annex.

A6.2.2.4 DEVICE MANAGEMENT

The BIS provides information to booting platforms about I/O units and I/O controllers. How the BIS gets its information is not specified, but attribute components that relate to I/O units and I/O controllers conform to the Device Management definition for components of the same name.

A6.2.3 CHARACTERISTICS

This section describes the characteristics of the BIS architecture.

Geographical Scope - There is no prescribed boundary for a BIS. Possibilities include:

- A BIS that serves all booting platforms of the entire fabric.
- A BIS that serves all booting platforms on particular subnets (e.g., a different BIS for each subnet).
- A BIS that serves booting platforms in particular partitions (e.g., a different BIS for each I/O partition).

Other possibilities exist and the various combinations are endless.

Redundancy - There may be more than one BIS serving the same set of booting platforms. A primary reason for multiple BIS entities is redundancy. See A6.4 “Booting using Boot Information Records” on page 1431 for recommendations on how a booting platform deals with more than one BIS.

How the BIS stores its boot information, the interface for configuring and updating that information, as well as the protocol for exchanging information with another BIS is outside the scope of this specification. Thus, this specification does not address coherency between redundant BIS servers nor any mechanism to provide redundant information.

A booting platform can not assume that multiple BIS servers all return the same information. There are other reasons for multiple BIS, such as different scopes (for example, a different BIS for each class of platform, for each processor type, or for each type of OS). When there are multiple BIS

servers with non-overlapping scope, all but one BIS returns zero records For example, a BIS for web servers would return zero records when queried by a print server, thus the print server would need to query other BISs until it found one that returns valid records. Of course, once the booting platform finds the information it needs, it no longer has a need for the BIS.

In addition to redundant BIS servers. A BIS can specify multiple devices even though the booting platform only needs a single device. A BIS provides information on devices in attributes called locator records, where each locator record identifies a device or service the booting platform is to use. When the BIS returns multiple locator records, it provides them in the order it wants the booting platform to use them.

BIS Availability - The BIS is a service for the multitude of booting platforms in the fabric. As such, at subnet initialization, booting platforms must wait for the BIS to become active. There are a number of factors that contribute to the latency of a BIS becoming available.

- The BIS must wait for the SM to configure switches, create path records, and set the BIS port's state to active;
- The platform that hosts the BIS must boot its OS and load the BIS application;
- The BIS must initialize and register with the SA as the BIS service.

These steps can take significant time and hence all booting platforms need to know how long to wait for the BIS. For this purpose, there is the PortBootInfo:BisTimeout value which can be set by default, set locally, or set by the BootManager.

Response Time - As per chapter 13, a BIS has to respond to a request within the timeout period specified in ClassPortInfo:RespTimeValue. The booting platform uses this timeout for determining if the request or response is lost. Thus, this time should indicate the worst case processing time for the BIS. However, there might be situations where the BIS can not respond immediately (for example, when a booting platform is not known to the BIS and the BIS needs to consult other sources before taking action).

The BIS may send KeepAlive packets (see A6.3.4.2.2 “Keep Alive Packets” on page 1417) until it determines how to respond. If the BIS has the capability to respond with KeepAlive packets, it should provide a means for the system administrator to disable that feature or limit the time that the BIS waits (continues sending KeepAlive) before it responds to the booting platform.

The KeepAlive packets inform the booting platform that the BIS needs more time before sending a final response. A booting platform is not re-

quired to wait for a final response. For instance, the booting platform can try another BIS or alternate resolution method.

Restricting Information - A BIS should not provide information to a booting platform about another node (such as an I/O unit) for which those two nodes do not share a common partition. Such information would be useless and it is undesirable for nodes to know about other nodes outside their assigned partitions. One way to accomplish this is for the BIS to validate that a path exists by querying the SA for path records between those two nodes. However this query will only work if the BIS shares a partition with each node for which it is requesting the path (they do not need to be the same partition). A BIS might periodically perform the validation or register with the SA to be informed if partition assignments change.

A6.3 BIS CLASS SPECIFICATION

This section specifies the Boot Information Service management class methods and attributes, operating requirements for a BIS, and interaction between a BIS and the subnet manager's subnet administration agent (SA). The various BIS relationships are illustrated in Figure 290.

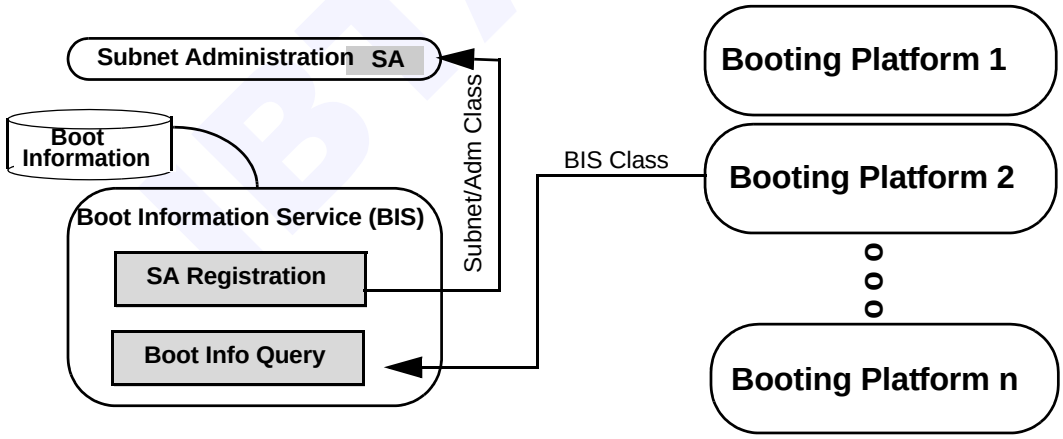


Figure 290 BIS Components & Interfaces

CA6-1: A BIS is a management agent and shall conform to the requirements for the Application Specific Management Agent (AMA) qualifier of the Optional Management Agent (OMA) compliance category as specified in section 20.12 Optional Management Agent Compliance Category on page 1116 and the Common MAD Header compliance category requirements specified in section 20.14 Common MAD Requirements on page 1119.

In particular, this requires conformance to the MAD format and use as specified in 13.4 Management Datagrams.

A6.3.1 REGISTRATION

BIS is a service for distributing boot information. A BIS may be implemented on any IBA node. The BIS registers as a subnet service with the SA using SubnAdmSet(ServiceRecord).

CA6-2: The BIS shall register with the SA via the SubnAdmSet(ServiceRecord) using the Null ServiceID (0x0000_0000_0000_0000) and <ServiceName>=”BIS.IBTA”, null terminated (i.e., the 9th and subsequent characters are 0x00). If the BIS is configured for multiple partitions, it shall register once for each partition.

The values for the other ServiceRecord parameters are implementation dependent.

For the BIS to register with the SA requires that the BIS supply the appropriate ServiceKey at the time it registers, updates, or modifies the ServiceRecord (see Chapter 15). The ServiceKey is the means for the SA to authenticate the BIS and protect against a rogue or invalid BIS. How a BIS gets configured with its ServiceKey is an implementation detail outside the scope of this annex. It is very important that a BIS takes appropriate steps to protect its service key and guard it such that other programs can not discover its value.

Normally, a BIS will not receive requests until after it has registered. How a BIS responds if it receives requests before it registers is undefined.

The BIS is responsible for periodically renewing its lease with the SA to prevent the SA from dropping its ServiceRecord from the SA database (see 15.2.5.14 ServiceRecord on page 895, and “15.2.5.14.3 ServiceLease on page 898).

CA6-3: The BIS shall renew its registration lease by reregistering with the SA before its service lease expires.

A booting platform queries the SA to get the ServiceRecord of the BIS and uses the ServiceGID from the ServiceRecord to query the SA for PathRecords to the BIS. Booting platforms ignore the ServiceID in the ServiceRecord because it does not need to resolve the ServiceID to a QP. The booting platform initially sends BIS class MADs to the GSI (i.e., QP1) of the port identified by the ServiceGID. If the BIS uses a different QP, then the GSI redirects the MAD to that QP.

A6.3.2 BIS QUERY OPERATION

The query methods (see A6.3.4.2 “Query Methods” on page 1415) permit a booting platform to query the BIS for boot information records. The content of the response depends on the source of the query and the content

of the query, that is, different booting platforms receive different information and the booting platform asks for a particular type of information.

A booting platform makes a query by sending a BisQuery() to the BIS. The query provides information identifying the booting platform, the capabilities of the booting platform, and the information the booting platform desires. A booting platform will query the BIS several times if it needs more than one type of information. For example, once to find a ROM Repository to extend its boot environment, then to locate a console, again to locate its boot device, and finally to locate a ROM Repository containing device drivers for the boot device.

The BIS uses the information provided in the query to generate a BisQueryResp() with boot information specific for that particular platform. Which components of the query that the BIS chooses for its filter criteria is implementation dependent, except that the BIS responds with zero or more records of the type requested by the booting platform.

The query methods permit the requester to submit a single query and the BIS to respond with a set of boot information records. The response may require more capacity for data than that provided by a single MAD. Thus the BIS query response uses the Reliable Multi-Packet Protocol (RMPP) specified in Chapter 13 for segmentation and reassembly of datastreams.

The attributes obtained by a BIS class query are records of boot information. Each boot information record is exactly 192 bytes, so each response MAD packet contains exactly one record. The BIS may return more than one record, all of the same type, and the AttributeID in the MAD header specifies which record type is being returned. Thus, each packet of a multi-packet response specifies the same attribute type. An example is a BIS returning four records identifying four boot devices. If the booting platform needs console and boot devices, it makes two queries, one for console and one for boot devices.

When a BIS responds with multiple Boot Information Records, it provides records in the order of preference (i.e., higher preference returned before lesser preference).

The BIS identifies a booting platform by the BootPlatformUUID component in the BisQuery(BootQueryInfo). When a platform is first introduced into the fabric, the BIS may not have locator records associated with the booting platform's BootPlatformUUID and might respond with (a) a status indicating no matching records, (b) a response that starts an install process, or (c) a Keep-Alive status while it acquires the information. After the BIS has been configured, the BIS is able to respond to a BisQuery() with a BisQueryResp() containing appropriate locator records. From this point forward, the BIS associates that BootPlatformUUID with that platform's locator records.

A6.3.3 BIS DATA FORMATS

Figure 291 shows the structure of the BIS management datagram. The datagram conforms to the MAD definition as specified in Chapter 13.

Bytes	Components
0 - 23	Common MAD Header (24 bytes as per 13.4.2 Management Datagram Format on page 718)
24-35	RMPP Header (12 bytes as per 13.6.2.1 RMPP Header on page 772)
36-63	reserved
64-255	AttributeData (192 bytes)

Figure 291 BIS MAD Structure

CA6-4: A BIS shall use the format specified in section A6.3.3 when sending and receiving BIS class MADs.

Table 410 defines the components of the BIS class datagram.

Table 410 BIS MAD Components

Component	Offset (Bytes)	Length	Description
BaseVersion	0	8 bits	Version of management datagram base format. Value is set to 1
MgmtClass	1	8 bits	Management class - set to 0x12 for BIS class
ClassVersion	2	8 bits	Version field. Value is set to 1.
BisMethod	3	8 bits	Method and R-Bit, as defined in A6.3.4 BIS Methods on page 1415.
Status	4	16 bits	see A6.3.3.2 BIS Status Values on page 1414
ClassSpecific	6	16 bits	not used. Shall be set to 0.
TransactionID	8	64 bits	Transaction specific identifier as per base MAD definition in 13.4.6.4 TransactionID usage on page 731
AttributeID	16	16 bits	Specifies the attribute in the AttributeData or the attribute to include in the reply, as per the Table 413 on page 1417 and Table 414 on page 1418.
reserved	18	16 bits	reserved.
AttributeModifier	20	32 bits	Identifies a particular instance of an attribute as per Table 413 on page 1417.

Table 410 BIS MAD Components (Continued)

RMPP_Header	24	12 bytes	Reliable Multi-Packet Protocol header as per 13.6.2.1 RMPP Header on page 772.
reserved	36	28 bytes	reserved
AttributeData	64	192 bytes	Data field used for operations as specified by Attribute definitions.

A6.3.3.1 RESERVED FIELDS

Chapter 13 requires components indicated as “reserved” to be set to zero in request messages. In response messages, a reserved field is either left unmodified if using the request for the response, or is set to zero. The recipient of a MAD ignores reserved fields.

A6.3.3.2 BIS STATUS VALUES

If the BIS rejects a request because it is too busy, it may respond with the appropriate response method setting MAD_Header:Status bit 0.

Bits 8-15 of the MAD_Header:Status field provide class specific definition for additional status encoded as follows:

Table 411 BIS MAD Status Field Components

Component Name	Bits	Meaning
BIS_Status	8-11	An enumerated value as follows: 0 = normal 1 = No records matching query 2 = Multi-packet request (RMPPType>0) invalid all other values reserved
reserved	12-15	reserved

CA6-5: If there are no records for a BisQueryResp(), the BIS shall set the MAD_Header:Status.BIS_Status in the BisQueryResp() to indicate no matching records and the AttributeData is undefined.

CA6-6: If the BIS receives a request (R bit = 0) with a ClassVersion other than 1, it shall respond by returning the request MAD with the R bit set to 1 and Status bits 2-4 set to the value of 1 (001b).

CA6-7: If the BIS receives a request with an unknown Method, it shall respond by returning the request MAD with the R bit set to 1 and setting Status bits 2-4 to the value of 2 (010b).

CA6-8: If the BIS receives a request with an unsupported Method/attribute combination, it shall respond by returning the request MAD with the R bit set to 1 and setting Status bits 2-4 to the value of 3 (011b).

CA6-9: If the BIS receives a request with an invalid value in any class specific MAD header component or any attribute component, it shall respond by returning the request MAD with the R bit set to 1 and setting Status bits 2-4 to the value of 7 (111b).

A6.3.4 BIS METHODS

CA6-10: A BIS shall support the BIS methods described in Table 412.

Table 412 BIS Methods

Method Name	Type[a]	Value	Description
BisGet()	S	0x01	Request (read) an attribute.
BisSet()	S	0x02	Request a set (write) of an attribute. This method is only used if the BIS supports vender traps or vender specific attributes.
BisGetResp()	S	0x81	The response from a BisGet() or BisSet().
BisQuery()	S	0x12	Requests (reads) all of the instances of the specified attribute that match the information supplied in the request.
BisQueryResp()	M	0x92	The response from a BisQuery() request using the Reliable Multi-Packet Protocol.

a. The “Type” column specifies if the method support multiple packets as follows:
[S] supports only single packets
[M] supports multiple packets using the Reliable Multi-Packet Protocol (see A6.3.4.2.1 “Multi-Packet Transaction” on page 1416).

A6.3.4.1 COMMON METHODS

BisGet(), BisSet(), and BisGetResp() methods are common methods (Get, Set, GetResp) specified in Chapter 13.

For these methods, the RMPP components (i.e., bytes 28-35) of the MAD are not used and thus filled with zeroes.

A6.3.4.2 QUERY METHODS

The BisQuery() and BisQueryResp() methods provide a mechanism for a booting platform to query the BIS for boot information records. The Query operation is illustrated in Figure 292. The content of the attribute that the booting platform supplies in the query request determines which attribute type the BIS returns in the query responses.

The BisQuery() is a single packet. Therefore, the RMPP_Header components (i.e., bytes 28-35) of the MAD are not used and thus filled with ze-

roes. The QueryResp() method uses the Reliable Multi-Packet Protocol (RMPP) defined in Section 13.6 Reliable Multi-Packet Transaction Protocol on page 770 and further specified in A6.3.4.2.1 Multi-Packet Transaction on page 1416.

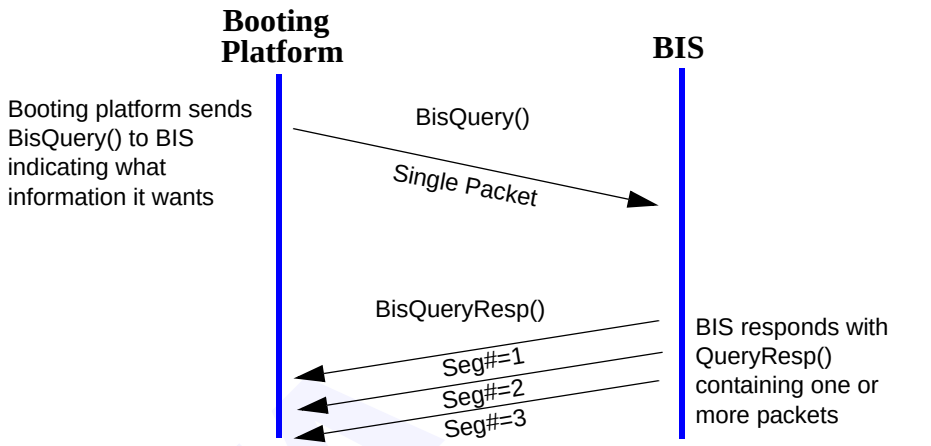


Figure 292 Query Operation

The attributes for the BisQueryResp() method are padded to 192 bytes such that each MAD provides one attribute record (i.e., one boot information record).

A6.3.4.2.1 MULTI-PACKET TRANSACTION

The BIS QueryResp method uses the Reliable Multi-Packet Protocol specified in 13.6 Reliable Multi-Packet Transaction Protocol on page 770 with the following class specific requirements.

CA6-11: A BIS shall implement the Reliable Multi-Packet Protocol for BisQueryResp() as per 13.6 Reliable Multi-Packet Transaction Protocol on page 770.

CA6-12: The BIS shall not send more than “Window component” number of packets before it receives an Ack from the requester. The Window value in each ACK determines the maximum number of subsequent packets the BIS is allowed to send.

CA6-13: If the BIS receives a BisGet(), BisSet(), or BisQuery() request indicating a multi-packet request (i.e., RMPPType not zero), then the BIS shall respond by returning the MAD with the R bit set to 1 and setting BIS_Status bits (MAD_Header:Status bits 8::11) to the value of 2 (Multi-packet request invalid).

A6.3.4.2.2 KEEP ALIVE PACKETS

The KeepAlive value in the RMPPType component identifies a KeepAlive packet.

Operation is as specified in 13.6 Reliable Multi-Packet Transaction Protocol on page 770. The BIS sets this value in a query response to request the recipient re-initialize its transaction response timer (see A6.3.4.3: Lost Messages).

AttributeData in a KeepAlive packet is indeterminate.

A6.3.4.3 LOST MESSAGES

The *ClassPortInfo* attribute provides a time-out value (see 13.4.8.1 ClassPortInfo on page 734) used in conjunction with BIS requests. Because BIS messages use unreliable datagram service, it is possible that a request or its response might be lost. If the querying node does not receive a response within the time specified, it resends the original request.

Where a response is a multi-packet segmented response, the segments are sent in order and identified by the SegmentNumber field. If the initiator detects that it missed one of the response packets, it can immediately request retransmission starting with the missing packet by sending a ReSend request (ReSend value in the RMPPType field), specifying the missing packet in the SegmentNumber field. The BIS starts resending the response starting with the indicated packet.

A6.3.5 ATTRIBUTES

Table 413 BIS Attributes

Attribute Name	Attribute ID	Attribute Modifier	Description
ClassPortInfo	0x0001	0x00000000	Class Info as per base MAD definition. This annex specifies BIS ClassVersion =1
BootQueryInfo	0x0080	0x00000000	Specifies information the BIS uses for Boot Information filter criteria
PlatformBootInfo	0x0020	0x00000000	Specifies platform-wide booting parameters
PortBootInfo	0x0021	0x00000000	Specifies port specific booting parameters
RomRepositoryLocator-Record	0x0030	0x00000000	Identifies an I/O unit that contains a ROM Repository
ConsoleLocatorRecord	0x0031	0x00000000	Supplies information describing a console service
OsLocatorRecord	0x0032	0x00000000	Supplies information describing a boot loader

Table 414 associates BIS attributes with methods.

CA6-14: A BIS shall support BIS method/attributes combinations identified in Table 414.

Table 414 BIS Attribute / Method Map

Method: / Attribute	Get / GetResp	Set / GetResp	Query	Query Resp
ClassPortInfo	x	x		
BootQueryInfo			x	
PlatformBootInfo				x
PortBootInfo				x
RomRepositoryLocatorRecord				x
ConsoleLocatorRecord				x
OsLocatorRecord				x

A6.3.5.1 CLASSPORTINFO

The ClassPortInfo attribute as defined in Chapter 13 has the following class specific information and definitions:

- ClassVersion: This is version 1 of the BIS class specification. The ClassVersion field shall be set to 0x01.
- CapabilityMask 8-15 = reserved, set to 0 and ignored

A6.3.5.2 BOOTQUERYINFO ATTRIBUTE

The BootQueryInfo attribute is used in a *BisQuery*() and supplies information needed for the BIS to filter its boot information and produce boot information records. The BootQueryInfo attribute has the format specified in Table 415.

Table 415 BootQueryInfo Attribute

Component Name	Offset (bits)	Length	Description
BootPlatformUUID	0	128 bits	A 128-bit universally unique identifier (UUID) as defined by ISO/IEC 11578 that uniquely identifies the booting platform.
PortGUID	128	64 bits	Port GUID of booting platform's port for which the booting platform is requesting information (see considerations below and section A6.3.5.4: PortBootInfo Attribute)
reserved	192	64 bits	reserved

Table 415 BootQueryInfo Attribute (Continued)

Component Name	Offset (bits)	Length	Description
BootSupport	256	32 bits	Identifies the capability of the booting platform. Bit specific: 0=no support, 1=supported. • **bit 0 - Extend Boot Environment** - boot environment can load code from a ROM Repository to extend the boot environment and/or BootAgent capability. • **bit 1 - Proprietary Driver Loa**d: the boot environment can load proprietary device drivers from a ROM Repository. • **bit 2 - IB Console Protocol** - The boot environment supports the IB Console protocol. This implies that the booting platform does not need to load (or already has loaded) an IB Console device driver from a ROM Repository. • **bit 3 - Proprietary Console Protocol** - The boot environment supports a proprietary console protocol. If bit 1 is set, the platform can load a console driver (including an IB Console driver). If bit 1is not set, this bit means that the booting platform has another source for console protocols. • **bit 4-SRP Storage Protoco**l- The boot environment supports the SRP protocol. This implies that the booting platform does not need to load (or already has loaded) an SRP device driver from a ROM Repository. • **bit 5 - Proprietary Storage Protocol** - The boot environment supports a proprietary storage protocol. If bit 1 is set, the platform can load a storage driver (including an SRP driver). If bit 1is not set, this bit means that the booting platform has another source for storage protocols. • **bit 6 - Network Boot: IB Network Model** - The boot environment supports a network boot protocol without using a LAN NIC (e.g., using IPoIB). • **bit 7 - Network Boot: IB NIC Model** - The boot environment supports network boot using a LAN NIC attached to the IB fabric. • **all other bits reserved**.
BootInfoRequested	288	8 bits	Indicates what boot information the booting platform wants. • 0x00 - **PlatformBootInfo** • 0x01 - **PortBootInfo** • 0x02 - **RomRepositoryLocatorRecords** to extend the boot environment • 0x03 - **RomRepositoryLocatorRecords** for device drivers • 0x04 - **ConsoleLocatorRecords** • 0x10 - **Install Program** Boot platform is requesting OsLocatorRecords describing the source for an OS installation program. This permits a booting platform to explicitly request an install. • 0x11 - **Boot Loader** Boot platform is requesting OsLocatorRecords describing the source of a boot loader. The BIS returns zero or more OsLocatorRecords, each describing the source for an OS boot loader or the source of an installation program. • 0x12 - **Boot Loader Destination** Boot platform is requesting OsLocatorRecords describing the destination for an OS boot loader (typically used by an installation program to determine where to install the OS boot loader). all other values reserved

Table 415 BootQueryInfo Attribute (Continued)

Component Name	Offset (bits)	Length	Description
reserved	296	24 bits	reserved
PlatformInfo	320	1024 bits (128-Bytes)	A TLV encoded component containing multiple packed elements as described in A6.3.5.2.5 “PlatformInfo” on page 1422
reserved	1344	192 bits (24 Bytes)	reserved

A6.3.5.2.1 BOOTINFOREQUESTED

The BIS returns the attribute specified in the BootInfoRequested component. For a PlatformBootInfo query, the BIS returns at most one PlatformBootInfo record. For a PortBootInfo query, the BIS returns at most one PortBootInfo record for the port identified in the PortGUID component. For the other BootInfoRequested types, the BIS might return multiple boot information records, one for each device or service for that particular request. Each Query Response packet of a multi-packet response contains one boot information record. For example, if there are 2 consoles and 3 storage devices that the BIS will provide to the booting platform, a 0x04 - “ConsoleLocatorRecord” query would return 2 MADs each containing a ConsoleLocatorRecord attribute and a 0x11 - “Boot Loader” query would return 3 MADs each containing an OsLocatorRecord attribute.

When trying to load its OS, a booting platform typically requests 0x11-“Boot Loader” and the BIS returns zero or more OsLocatorRecords that identify an OS boot loader or a boot loader that loads an installation program. If the BIS knows that this in a new platform and wants to start the install process, then it can return one or more OsLocatorRecords identifying an Install Program. Otherwise, the BIS would return one or more OsLocatorRecords that each describe a source for the platform’s OS Boot Loader. The BIS may choose to return combinations of OsLocatorRecords (both OS boot loader source and installation program source). For example, the BIS might return two OsLocatorRecords, the first describing a disk drive and the second describing the source of an OS Install Program. If the booting platform detects a failure in the first source, such as failing to find a valid boot sector on the disk or detects that the OS was not installed, then it proceeds to the next OsLocatorRecord and thus installs the OS. If it does find a valid OS boot loader, i.e., the installation was successful, then the booting platform boots the OS from the first OsLocatorRecord and does not use the second.

A booting platform can specify a BootInfoRequested component of 0x10 - “Install Program” when it knows it wants to perform an OS install. During the installation, the Install Program can query the BIS, specifying a BootIn-

foRequested component of 0x12 - “Boot Loader Destination”, to get a list of destinations where the Install Program can place the OS Boot Loader.

A point of clarification concerning the use of RomRepositoryLocator-Records: There are two phases in extending the capabilities of the booting platform and thus two encodes for requesting RomRepositoryLocator-Records.

- One phase is in extending the boot environment, such as loading updated or extended code. It is possible that a platform only contains a minimal amount of boot code, which simply locates a ROM Repository and loads the remainder of the boot environment from it. In this case, the booting platform is looking for platform specific or boot environment specific code stored in the ROM Repository. This phase occurs prior to the booting platform requesting any OsLocatorRecords.
- The second phase is when the booting platform is processing a locator record and that record identifies an I/O protocol for which the booting platform needs to load a driver. In this case, the platform is looking for a specific I/O driver. Refer to the I/O Annex for driver matching rules.

To permit platform code and device driver code to reside in different ROM Repositories, there are two encodes to indicate for which phase the booting platform is requesting a ROM Repository.

A6.3.5.2.2 BOOTPLATFORMUUID

CA6-15: A BIS shall only return attributes for the platform identified by BootPlatformUUID component.

How a BIS manages its boot information and filters it is outside the scope of this annex. However, an implementer might want to consider the following issues.

The BIS identifies a booting platform by its BootPlatformUUID. The 128-bit UUID was selected over a 64-bit GUID (EUI-64) because it can be generated easily by software, which decouples the boot environment from any particular hardware. For example, it removes the necessity for a hardware serial number to be machine readable. Software generation is also advantageous for retrofit and field upgrades in machines that are not specifically designed for InfiniBand. A BIS does not need to interpret the BootPlatformUUID, but rather uses it as an opaque value to match with locator records assigned to that BootPlatformUUID.

As a matter of policy, a BIS may associate the BootPlatformUUID with a set of physical GUIDs (such as Node GUIDs, Port GUIDs, Module GUIDs, etc.) and validate with the SA that the platform making the request has the same physical attributes each time it receives a query. Detecting a different physical GUID indicates either the requesting platform has physi-

cally changed, or that the platform is not who it claims to be. How the BIS determines which is the case, and what actions the BIS takes is an implementation policy.

A6.3.5.2.3 PORTGUID

When a BIS returns locator records, they identify devices that the booting platform needs, independent of which port the booting platform uses to query the BIS. That is, a booting platform attempts to locate the device through each of its ports, until it finds a suitable path. However, when the target has multiple ports, the PortGID that the BIS returns in the locator record could be a function of the PortGUID component that the booting platform specifies in the BisQuery(). The expectation is that the booting platform identifies the first port it intends to use and, unless the BIS has specific topology information, the BIS ignores the PortGUID component.

When the booting platform requests PortBootInfo, the BIS returns the attribute for the booting platform' s port specified in the PortGUID component of the BisQuery(). The BIS responds with a MAD_Header:Status.BIS_Status = 1:"No records matching query" when it does not have the information for the specified port. When the BIS does return a PortBootInfo record, the record's RomPortPriority, ConsolePortPriority, IocPortPriority, and NetworkBootPortPriority components indicate if and how the booting platform uses the specified port for booting. The Booting Annex describes how the booting platform uses those priorities.

A6.3.5.2.4 BOOTSUPPORT

The BootQueryInfo:BootSupport component is provided as supplemental information and there is no requirement that BIS use that information. It could be useful when the BIS does not recognize the UUID. The BIS may use this information in determining the status and content of the BisQueryResp().

A6.3.5.2.5 PLATFORMINFO

The BootQueryInfo:PlatformInfo component specifies booting related characteristics of the booting platform. This information is useful when the booting platform is unknown to the BIS, such as when a new platform is added to the fabric, and thus the BIS can not associate the booting platform with a prior set of locator records.

The PlatformInfo component contains multiple elements, where each element is a platform characteristic (see Table 416, "PlatformInfo Elements," on page 1424). The elements in PlatformInfo allow the BIS to identify the booting platform to the system administrator or automatically invoke an install process based on the content of the elements. Other possibilities include using the information to validate the requestor and for reporting changes to the System Administrator.

The BootQueryInfo:BootPlatformUUID is the primary component that a BIS uses to identify a booting platform and the PlatformInfo component is intended as supplemental information to identify those characteristics of a booting platform that may influence the set of locator records selected for the booting platform.

Elements in PlatformInfo are in TLV format. The first byte is the Type, the second byte is the Length (number of bytes in the value string), and the remainder of the element is the value string, in UTF-8 format. Type codes are specified in Table 416. The PlatformInfo component contains a variable number of variable length elements and is terminated with the null value (0x00). All bytes after the termination byte should also be null bytes.

Each respective booting platform vendor is responsible for defining its own value string definition. For instance the Platform vendor defines unique values for each of its products and firmware vendor defines unique values for each of its products. An element with a Length of zero means that the booting platform does not know the value for that element.

The recommended practice is for the booting platform to order elements by their Type value, lowest to highest.

Table 416 PlatformInfo Elements

Type Value	Length Value	Description (All strings are in UTF-8 and the content of each element is vendor specific)
0x00	0x00	Marks end of elements - ignore remainder of component data following this Type code
0x01	variable	**Platform Vendor Name String** This string provides the name of the vendor that manufactured the booting platform.
0x02	variable	**Platform Vendor Model/Type String** This string provides the model and type of the booting platform.
0x03	variable	**Platform Serial Number String** This string provides the name of the serial number of the booting platform.
0x04	variable	**Firmware Vendor Name String** This string provides the name of the firmware vendor that manufactured the boot environment code (e.g., BIOS vendor.).
0x05	variable	**Firmware Version String** This string provides the version of the firmware code.
0x06	variable	**CPU Vendor Name String** This string provides the name of the CPU vendor
0x07	variable	**CPU Vendor Version String** This string provides the CPU version, stepping, etc.
0x08	variable	**Platform Name String** This string provides the local name assigned the booting platform - usually assigned by the System Administrator to identify the platform by name.
0x09	variable	**OS Name String** This string provides the name of the preferred Operating System.

A6.3.5.3 PLATFORMBOOTINFO ATTRIBUTE

The PlatformBootInfo attribute is a subset of the R/W components of the BootMgt:PlatformBootInfo attribute (see Booting Annex) and supplies the booting platform with platform-wide booting parameters.

Table 417 PLATFORMBOOTINFO Attribute

Component Name	Offset (bits)	Length	Description
reserved	0	36-bits	reserved
RomRepositoryLocator-Source	36	4-bits	This component indicates how and in what order the booting platform locates a ROM Repository to extend its boot environment. • 0x0 - BIS Only • 0x1 - BIS then Persistent • 0x2 - Persistent Only • 0x3 - Persistent then BIS • 0xF - No ROM Repository for expansion.
ConsoleLocatorSource	40	4-bits	This component indicates how and in what order the booting platform locates a Console. • 0x0 - BIS Only • 0x1 - BIS then Persistent • 0x2 - Persistent Only • 0x3 - Persistent then BIS • 0xF - No Console.
OsLocatorSource	44	4-bits	This component indicates how and in what order the booting platform locates an IOC, device or server containing its operating system boot loader storage and network booting: • 0x0 - BIS Only • 0x1 - BIS then Persistent • 0x2 - Persistent Only • 0x3 - Persistent then BIS • 0xF - Disable/No IB boot}
reserved	48	186-Bytes	reserved

CA6-16: A BIS shall respond to a BisQuery(BootQueryInfo:BootInfoRequested=0x00 - PlatformBootInfo) with a BisQueryResp(PlatformBootInfo) containing at most one PlatformBootInfo record as specified in Table 417.

A6.3.5.4 PORTBOOTINFO ATTRIBUTE

The PortBootInfo attribute is identical to the R/W components of the BootMgt:PortBootInfo attribute and supplies the booting platform with port-specific booting parameters for the port identified in the query.

Table 418 PORTBOOTINFO Attribute

Component Name	Offset (bits)	Length	Description
BISPortPriority	0	2-bits	This component indicates the port's priority for locating a BIS. • 11b is the highest priority • 01b is the lowest priority • 00b indicates that the port should not be used to locate a BIS.
RomPortPriority	2	2-bits	This component indicates the priority of this port when attempting to locate a ROM repository. • 11b is the highest priority • 01b is the lowest priority • 00b indicates that the port should not be used to locate a ROM repository.
ConsolePortPriority	4	2-bits	This component indicates the priority of this port when attempting to locate a console. • 11b is the highest priority • 01b is the lowest priority • 00b indicates that the port should not be used to locate a console.
IocPortPriority	6	2-bits	This component indicates the priority of this port when attempting to locate an I/O unit (I/O Controller). • 11b is the highest priority • 01b is the lowest priority • 00b indicates that the port should not be used to locate an I/O unit.
NetworkBootPortPriority	8	2-bits	This component indicates the priority of this port when attempting an IB network boot. • 11b is the highest priority • 01b is the lowest priority • 00b indicates that the port should not be used for IB Network Boot.
reserved	10	22-bits	reserved
InitTimeout	32	16-bits	The time (100 mSec increments) from Power On Reset that the booting platform allows for subnet resources (e.g., I/O units) to become operational.
BisTimeout	48	16-bits	The time (100 mSec increments) from Power On Reset that the booting platform allows for the BIS to become operational.
EndNodeTimeout	64	16-bits	Specifies the maximum amount of time (in 100 mSec increments) that an end node takes to become operational and thus respond to MADs. Timer starts when the booting platform first sends a MAD to the particular node. This component accounts for target nodes that are in a power-down state and are awakened as a result of the booting platform sending MADs to it.
reserved	80	182-Bytes	reserved

CA6-17: A BIS shall respond to a BisQuery(BootQueryInfo:BootInfoRequested=0x01 - PortBootInfo) with a BisQueryResp(PortBootInfo) containing at most one PortBootInfo record as specified in Table 418.

A6.3.5.5 ROMREPOSITORYLOCATORRECORD ATTRIBUTE

The RomRepositoryLocatorRecord attribute used in the *BisQueryResp()* provides a boot information record identifying an I/O unit containing a ROM Repository and has the format specified in Table 419. The AttributeID in the MAD header indicates that the AttributeData contains a RomRepositoryLocatorRecord record.

The RomRepositoryLocatorRecord attribute is identical to the R/W components of the BootMgt:RomRepository attribute.

Table 419 ROMREPOSITORYLOCATORRECORD Attribute

Component Name	Offset (bits)	Length	Description
reserved	0	16-Bytes	reserved
PortGID	128	16-Bytes	Port GID of the I/O unit.
reserved	256	160-Bytes	reserved

CA6-18: A BIS shall respond to a BisQuery(BootQueryInfo:BootInfoRequested=0x02 - RomRepositoryLocatorRecords to extend the boot environment or 0x03 - RomRepositoryLocatorRecords for device drivers) with a BisQueryResp(RomRepositoryLocatorRecord) as specified in Table 419.

A6.3.5.6 CONSOLELOCATORRECORD ATTRIBUTE

The ConsoleLocatorRecord attribute used in the *BisQueryResp()* provides a boot information record for a console service and has the format specified in Table 420. The AttributeID in the MAD header indicates that the AttributeData contains a ConsoleLocatorRecord attribute. Note that IB Console Protocol (see Console Annex) supports both console server processes and console devices.

Table 420 ConsoleLocatorRecord Attribute

Component Name	Offset (bits)	Length	Description
reserved	0	8-bits	reserved
Device-Service	8	1-bit	Indicates if this record describes a Console IOC or Console Server Process • 0b = this record describes a console IOC • 1b = this record describes a console server process
DeviceDriverLocation	9	2-bits	Specifies the search order to locate a Device Driver for a Console IOC that supports a proprietary device driver. • 00b = Search only the ROM Repository in the same IOU as the IOC. • 01b = First search the ROM Repository in the same IOU as the IOC, then try ROM Repository records. • 10b = First search ROM Repositories pointed to by RomRepository records, then try the ROM Repository on the same IOU as the IOC. • 11b = Search only ROM Repositories pointed to by the RomRepository Records.
reserved	11	5-bits	reserved
Protocol	16	8-bits	The console protocol to use (see A6.3.5.8 Protocol Field on page 1430): • 0x00 - unknown otherwise bit specific where: • bit 0 - proprietary protocol • bit 1 - IBTA Console protocol (refer to Console Annex) • bits 2-6 are reserved • bit 7 -Use any ProtocolName specified by a ProtocolName element in the AdditionalInfo component.
reserved	24	5-Bytes	reserved
IocGUID-SID	64	8-Bytes	GUID of the I/O controller or ServiceID of the service depending on setting of the Device-Service field • If Device-Service = 0b (console IOC), then this is the IocGUID. • If Device-Service = 1b (server process), then this is the ServiceID. A service ID other than the IB CSP ServiceID indicates a proprietary console protocol
PortGID	128	16-Bytes	Port GID for the console service.
AdditionalInfo	256	160-Bytes	Console protocol specific data in Type-Length-Value (TLV) format see Booting Annex for definitions

CA6-19: A BIS shall respond to a BisQuery(BootQueryInfo:BootInfoRequested=0x04 - ConsoleLocatorRecords) with a BisQueryResp(ConsoleLocatorRecord) as specified in Table 420.

A6.3.5.7 OSLOCATORRECORD ATTRIBUTE

The OsLocatorRecord attribute used in the *BisQueryResp()* provides a boot information record for boot device such as a storage device or a network interface adapter, and has the format specified in Table 421. The AttributeID in the MAD header indicates that the AttributeData contains an OsLocatorRecord attribute.

Table 421 OsLocatorRecord Attribute

Component Name	Offset (bits)	Length	Description
BootMethod	0	8-bits	Boot Method • **0x01** = Storage (attribute specifies IOC to use) • **0x02** = Network (attribute specifies IOC to use) • **0x03** = Proprietary (attribute specifies IOC) • **0x04** = IB Network boot (attribute specifies IB boot server, if needed) all other values reserved
reserved	8	1-bit	reserved
DeviceDriverLocation	9	2-bits	Specifies the search order to locate a Device Driver for an IOC. • 00b = Search only the ROM Repository in the same IOU as the IOC. • 01b = First search the ROM Repository in the same IOU as the IOC, then try ROM Repository records. • 10b = First search ROM Repositories pointed to by RomRepository records, then try the ROM Repository on the same IOU as the IOC. • 11b = Search only ROM Repositories pointed to by the RomRepository Records.
RecordType	11	2-bit	Indicates the purpose of this record • 00b = this record describes a source for a boot loader that loads an installation program which can install an OS for the booting platform. • 01b = this record describes a source for the booting platform's OS boot loader • 10b = this record describes a destination where an installation program can install an OS boot loader. A booting platform should not attempt to use an OsLocatorRecord with RecordType=10b as the source for its boot loader. • 11b = this record describes a destination where an installation program can or has installed an OS boot loader. A booting platform can attempt to boot from this location. An installation program can install an OS boot loader at this location.
reserved	12	3-bits	reserved
Protocol	16	8-bits	The IOC protocol to use (see A6.3.5.8 Protocol Field on page 1430): • 0x00 - unknown otherwise bit specific as described in Table 422, "Protocol Component Bit Definitions," on page 1431)
reserved	24	5-Bytes	reserved

Table 421 OsLocatorRecord Attribute (Continued)

Component Name	Offset (bits)	Length	Description
IocGUID-SID	64	8-Bytes	GUID of the I/O controller or SID of the Boot Service (a SID value of zero means unknown or not used).
PortGID	128	16-Bytes	Port GID of the I/O unit or the Boot Service (a value of zero means unknown or not used).
AdditionalInfo	256	160-Bytes	I/O protocol specific data in Type-Length-Value (TLV) format, refer to Booting Annex for definition and content.

CA6-20: A BIS shall respond to a BisQuery(BootQueryInfo:BootInfoRequested=0x10 - Install Program) with a BisQueryResp(OsLocatorRecord) as specified in Table 421 and the RecordType component of any record returned shall be 00b.

CA6-21: A BIS shall respond to a BisQuery(BootQueryInfo:BootInfoRequested=0x11 - Boot Loader) with a BisQueryResp(OsLocatorRecord) as specified in Table 421 and the RecordType component of any record returned shall be 00b, 01b, or 11b.

CA6-22: A BIS shall respond to a BisQuery(BootQueryInfo:BootInfoRequested=0x12 - Boot Loader Destination) with a BisQueryResp(OsLocatorRecord) as specified in Table 421 and the RecordType component of any record returned shall be 10b or 11b.

A6.3.5.8 PROTOCOL FIELD

Locator records contains a Protocol component that the BIS uses to specify which protocols the booting platform can use with the specified device. This field is bit specific (i.e., each bit represents a protocol). When a device supports more than one protocol, the BIS can indicate a preference by returning multiple records, each with a different Protocol value. If the BIS sets more than one bit, it means that the booting platform may use any of the indicated protocols. A value of zero means that the BIS does not know which protocols the device supports. In this case the booting platform determines which protocol to use.

For OsLocatorRecords, Protocols are dependant on the BootMethod component as specified in are Table 422.

Table 422 Protocol Component Bit Definitions

Bit	Boot Method			
	01 Storage	02 LAN Network Boot	03 Proprietary	04 IB Network Boot
0	proprietary	proprietary	proprietary	proprietary
1	SRP	reserved	reserved	IPoIB
2-6	reserved			
7	Use any ProtocolName specified by a ProtocolName element in the AdditionalInfo component.			

When bit 7 is set, the booting platform looks in the AdditionalInfo component of the OSLocatorRecord for ProtocolName elements to determine which I/O protocols it can use.

A6.4 BOOTING USING BOOT INFORMATION RECORDS

This section specifies the recommended practices and considerations for a booting platform that uses a BIS to derive boot information. This annex only specifies requirements on the BIS. See the Booting Annex for requirements on a booting platform.

A6.4.1 OVERVIEW

The booting platform uses the services of the subnet manager's subnet administration agent to identify and locate a BIS and then queries the BIS for boot information. In each query the booting platform specifies which type of information it wants and can ask for one of the following:

- boot parameters for the platform (PlatformBootInfo);
- boot parameters for a particular port (PortBootInfo);
- location of ROM Repositories (RomRepositoryLocatorRecords) that contains code to extend the boot environment or that contain device drivers;
- the location of a console (ConsoleLocatorRecords);
- the location of a boot loader (OsLocatorRecords).

If a booting platform needs multiple types of information, it queries the BIS multiple times. For each query, the BIS returns the requested information.

The booting platform queries the BIS (as specified in A6.3.5.2 “BootQueryInfo Attribute” on page 1418) to get its boot information records. When responding to a query for the location of a ROM Repository, Console, or Boot Loader, the BIS may return multiple records using the Reliable Multi-Packet Protocol. The booting platform assumes the BIS returns the records in the order of priority (primary device/service first). Records are returned as a part of multi-packet response such that the booting platform can detect and request retransmission of missed records. The BIS might also respond with a KeepAlive status when it needs more time to process the query. This is useful for when the BIS needs to poll the fabric or query other data repositories.

A6.4.2 GENERAL OPERATION

The following is an example of how a booting platform can use the Boot Information Service stated in relative order.

1) Each BIS registers itself with the SA using the **SubnAdmSet(ServiceRecord)** method with *ServiceName* = “BIS.IBTA” null terminated (i.e., the 9th a subsequent characters are 0x00). The SA ServiceRecord lease will remove stale entries after the lease expires (see Chapter 15). However, a booting platform should be able to deal with a stale ServiceRecord for a BIS that is no longer in service (i.e., the SA might return zero path records from the booting platform to the BIS or the BIS does not respond).

2) Since each port of a booting platform might be on a different subnet and/or each port may belong to a different set of partitions, the booting platform should treat each of its ports as if it is independent from the others. Thus, the booting platform selects a port and all subsequent operations (i.e., the following steps) are performed through that port. Note that the Booting Annex provides the means for a booting platform to be configured with BIS port priorities that identify which order the booting platform attempts to use its ports to locate a BIS.

3) The booting platform uses **SubnAdmTableGet(ServiceRecord)** method to query the SA and retrieve a list of all the BIS agents (i.e., ServiceName = “BIS.IBTA“) for all of its partitions (i.e., wildcards the ServiceP_Key component in the request) and selects one. When the booting platform finds multiple BIS servers, it should attempt to use them in the order that the SA returns the Service Records.

4) The booting platform queries the SA to get the path record for the selected BIS using the **SubnAdmTableGet(PathRecord)** with DGID = the GID from the service record. From this information the booting platform selects a path and is now able to send messages to the GSI of the selected BIS.

5) The booting platform queries the BIS using the **BisQuery(BootQueryInfo)** message described in A6.3.4.2 Query Methods on page 1415. The booting platform makes a separate query (i.e., a single packet request) for each type of information it wants in any order it wants to request it. By setting the appropriate value in the BootInfoRequested component, the booting platform can request one of the following:

   a) A PlatformBootInfo record specifying platform booting parameters

   b) A PortBootInfo record specifying booting parameters for the specified port

   c) ConsoleLocatorRecords specifying a console

   d) RomRepositoryLocatorRecords for ROM Repositories for extending the booting platform's boot environment

   e) RomRepositoryLocatorRecords for ROM Repositories containing device drivers

   f) OsLocatorRecords for boot devices (storage, Network Interface Controllers, etc.) or Network Boot information that the booting platform uses to locate and load a boot loader.

6) The BIS may redirect a query to a dedicated QP, which might be on another port. The BIS does this by responding with a **BisGetResp(ClassPortInfo)** specifying the redirection information (see section 13.5 MAD Processing on page 749). For this case the booting platform resubmits the query to the redirected port. Redirection can come at anytime. All subsequent queries should be to the redirected port.

7) In certain cases, the BIS may need more time to process boot inquiries, and thus it responds with a KeepAlive status. This response is useful when the BIS information is stale or the BIS needs to validate the requestor and thus the BIS takes an action that requires additional time (such as querying a configuration manager, etc.). A booting platform is not required to wait and thus the amount of time it does wait is a policy of the booting platform.

8) The BIS eventually responds with boot information records for the particular request (or the platform times out and jumps to step 10).

9) The booting platform attempts booting using the appropriate protocols, loading proprietary drivers as necessary. The ports that the booting platform uses to access the ROM Repository, Console, and boot loader are determined by the appropriate PortBootInfo priority values.

10) If no suitable boot devices or services are found, the booting platform might take one or more of the following actions, not necessarily in this order:

a) Query the next BIS (return to step 4)

b) Repeat the process for the next port (return to step 3)

c) Periodically query the SA looking for a new BIS ServiceRecord (return to step 3) and repeat the process. If no new BIS, then retry each BIS. This covers the case of a new BIS registering and also a current BIS being updated with new information.

d) Report an error to the console

e) Send a Trap to the Boot Manager

f) Stall and await operator assistance

g) Continue with another boot resolution method (see Booting Annex).

A6.5 COMPLIANCE SUMMARY

This annex specifies a new Compliance Category (see Chapter 20: Volume 1 Compliance Summary on page 1072 for explanation of compliance categories and qualifiers). The new category is BIS-Server. There are no Compliance Qualifiers.

In order to claim compliance to the InfiniBand Architecture Specification for the Compliance Category of BIS-Server, a product shall meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support (currently there are no optional compliance qualifiers for this category).

In addition, a BIS must also be compliant with the general management framework requirements from Chapter 13 specified in Section 20.12, “Op-

tional Management Agent Compliance Category," on page 1116 and Section 20.14, "Common MAD Requirements," on page 1119.

ANNEX A7: CONFIGURATION MANAGEMENT

A7.1 INTRODUCTION

This annex and Annex A8: Device Management defines the framework for managing I/O units and assigning I/O Unit resources to client platforms. The Device Management annex specifies the Device Management class and the requirements for an I/O unit to support Device Management. This annex describes the overall configuration management framework, specifies the Device Administration Class, and specifies the requirements for a configuration management application that configures the I/O Unit (i.e., the Device Manager) and administers privileged configuration information to client platforms (i.e., Device Administrator). The combination of the Device Manager and Device Administrator is referred to as a Configuration Manager.

This Annex:

- Explains the framework for managing the resources of an I/O unit (i.e., I/O service objects) and assignment of those resources to hosts (i.e. client platforms). The overall objective is to facilitate multiple hosts sharing the same I/O unit. In this annex, a host that is assigned I/O resources is referred to as a client platform.
- Specifies the role of a configuration manager, which performs two main functions. It is a Device Manager (DM) that uses Device Management (DevMgt) class MADs (see Annex A8: Device Management) to configure I/O units and it is a Device Administrator (DA) that uses Device Administration (DevAdm) class MADs (specified in this annex) to provide hosts with I/O configuration information and to coordinate I/O events (such as diagnostics and hot plug),
- Defines the Device Administration management class, which specifies the management interface (messages, methods, and attributes) between Client Platforms and the DA.

I/O units are capable of providing I/O service to multiple clients. However, it is desirable to control which I/O resources each client is permitted to use. IB partitioning enforces isolation among systems sharing an InfiniBand fabric, but does not provide for partitioning of the resources within a node. Device Management in conjunction with Device Administration and Communication Management (CM; see chapter 12) together extend the IB Architecture to provide enforceable assignment of I/O resources to multiple clients.

The Device Management class provides the means to configure an I/O unit to specify which client platforms are permitted to access which I/O services (by associating access keys with a set of I/O service objects). The Device Administration class provides the means for client platforms to get information (such as access keys) that enables those platforms to use allocated services and to be notified about events that affect their use of those resources.

A7.1.1 GLOSSARY

The following terms are in addition to the terms in Volume 1 Chapter 2: Glossary on page 69.

Active Configuration Management — The usage model where there is a Device Manager that configures I/O Units with access rights for Service Clients and thus restricts which clients can see and use certain I/O resources. Also see Passive Configuration Management. Device Management in conjunction with Device Administration and Communication Management together provide enforceable assignment of I/O resources to multiple clients.

Active Device Management — Another term for Active Configuration Management.

Client_Key — A key that a client passes to the I/O Unit in CM and DevMgt MADs. The I/O Unit allows the client to access service objects based on the record in the Client Pool Table that matches this key.

Client Platform — A platform (one or more channel adapters under common control) that hosts one or more Service Clients. Also see Figure 293 "Configuration Management Usage Model" on page 1443.

Client Pool Table — A table in an I/O Unit that specifies the access rights of each client. Each record specifies the list of service objects that a client is allowed to access and specifies the I/O Unit CA resources, such as number of QPs, the client may consume. Each record is identified by a unique Client_Key and specifies the Supervisor_Key of the Client Platform that is permitted to modify the record.

Configuration Group — The set of I/O Units that are managed by the same Device Manager and the set of Client Platforms that are permitted to use those I/O Units.

Configuration Management — The act of configuring I/O Units to control which Service Clients may access which of the IOU's services and administering configuration information to client platforms. See Device Management and Device Administration.

Configuration Manager — A manager that provides the Device Manager and Device Administrator for a Configuration Group.

DA — Device Administrator

DevAdm — Device Administration class

Device Administration — An IB management class for administering information about I/O Units to Client Platforms.

Device Administrator — A function of the Configuration Manager that administers configuration information to Client Platforms.

Device Management — An IB management class for obtaining information about an I/O Unit's resources, configuring the I/O unit, and for managing diagnostics.

Device Management Agent — An IB management agent in an I/O Unit that implements the Device Management class protocol. It provides information about the I/O Unit's I/O resources, can invoke diagnostics, and reports diagnostic results.

Device Manager — An IB manager that configures an I/O Unit by setting the I/O unit's Device Management class attributes via the I/O Unit's Device Management Agent using DevMgt class MADs. As a class manager, it provides the means for 3rd parties to subscribe for DevMgt class event reports (i.e. Trap forwarding). It also oversees I/O module hot plug and diagnostic sessions.

DM — Device Manager

DevMgt — Device Management class

DevMgt Agent — Device Management Agent

Host — See Client Platform

IOC — I/O Controller

I/O Client — A function or process on a Client Platform that uses an I/O service provided by an I/O Unit (i.e., the user of one or more I/O Service Objects).

I/O Controller — A process or circuit of an I/O Unit that provides access to one or more I/O Service Objects.

I/O Device — An overloaded term that usually refers to an I/O Unit, an I/O Controller, an I/O Service Object, or a Protocol Object. In this annex it refers to any physical or logical I/O component that an I/O Controller accesses on behalf of an I/O Client, see Figure 308 "Model for an I/O Unit" on page 1515 of Annex A8: Device Management. An example of an I/O Device is a SCSI disk drive behind a SCSI I/O Controller.

I/O Management Application — A process running in a Client Platform[41] that accesses an I/O Management Object to create, configure, or destroy I/O Service Objects and/or set

I/O protocol-specific parameters. Also see Figure 309 "I/O Components and Relationships" on page 1517 of Annex A8: Device Management.

I/O Management Object

A type of Service Object provided by an I/O Unit used to configure I/O Service Objects and I/O Protocol-specific parameters. Not to be confused with an IB Management Agent, the DevMgt class advertises the existence of vendor supplied management objects that do not use the IB management framework (i.e., do not use GMPs and QP1). Also see Figure 309 "I/O Components and Relationships" on page 1517 of Annex A8: Device Management.

I/O Management Protocol

The protocol between an I/O Management Application and an I/O Management Object, typically to manage I/O configuration such as installing I/O Controllers or creating, destroying, and configuring Service Objects. I/O Management Protocols are outside the scope of this annex, except that DevMgt advertises the I/O Management Protocols supported by an I/O Unit and its I/O Controllers.

I/O Module

A permanent or removable subassembly of an I/O Unit that contains one or more I/O Controllers. An I/O module is different from an IB module in that it does not contain the channel adapter.

I/O Partition

Any partition used to perform I/O. An I/O partition is defined as the set of Client Platforms and I/O Units that share the same P_Key value for the purpose of supporting I/O operation. An I/O partition exists by virtue of the SM assigning to an I/O Unit, a P_Key value that permits a Client Platform to access that I/O Unit. IBA requires a Client Platform to be a member of one of the I/O Unit's I/O partitions in order to use the I/O Unit.

I/O Port

An overloaded term, in this annex it refers to a physical attachment to an I/O fabric, for example a Fibre Channel port.

I/O Protocol

The protocol between an I/O Client and an I/O Service Object to invoke I/O operation and perform I/O transactions. I/O Protocols are outside the scope of this annex, except that DevMgt identifies the I/O Protocols supported by an I/O Unit (i.e., its I/O Controllers).

IORM

I/O Resource Manager

I/O Resource Manager

A management function in a Client Platform that manages I/O resources for that platform. Also known as the platform's Supervisor. The Device Administrator provides each IORM with a Supervisor_Key. The IOU allows

41. Typically, management applications run from an administrator's machine. For the purpose of accessing the I/O management object in the IOU, the machine is considered a client because Device Management provides the means for the Device Manager to configure which nodes may access specific I/O management objects.

	the IORM access to DevMgt attributes based on the records that matches this key. An IORM uses the key to read its Platform Pool Table record and to modify the Client Pool Table records assigned to that platform to further control which IOU resources each of its clients may access.
I/O Service Object	A type of Service Object provided by an I/O Controller that permits an I/O Client to access a set of I/O functions or I/O Devices via an I/O Protocol (see Figure 309 "I/O Components and Relationships" on page 1517 of Annex A8: Device Management). Each I/O Service Object uses a different set of QPs. That is, an I/O client uses a different channel to access each service object.
IOU	I/O Unit
I/O Unit	A node that implements a Device Management Agent. It normally contains one or more I/O Controllers and provides I/O service.
Passive Configuration Management	A usage model where there is not a Device Manager that configures I/O Units. Management uses only partitions (P_Keys) to control which nodes may access an I/O unit and thus see and use its I/O resources. For passive management, if a node has access to an I/O Unit, then any I/O Client on that node has access to all I/O resources provided by that I/O unit. Also see Active Configuration Management.
Passive Device Management	Another term for Passive Configuration Management.
Platform Pool Table	A table of records in an I/O Unit where each record specifies the access rights of a Client Platform (i.e., list of service objects that each platform is allowed to access and specifies the I/O Unit CA resources, such as number of QPs, the platform may consume). Each record is identified by a unique Supervisor_Key.
Protocol Object	A logical I/O entity that is accessible through an I/O Service Object. The means to discover and address a protocol object is I/O Protocol specific. An example of a Protocol Object is a SCSI logical unit, which is distinguished by its SCSI logical unit number (LUN) in the SCSI protocol packet. Protocol Objects are outside the scope of Devmgt. That is, I/O clients use DevMgt to identify I/O Service Objects and then communicate with each I/O Service Object via its I/O Protocol to identify and address Protocol Objects.
Service Client	An entity such as an I/O Client or I/O Management Application that communicates with a Service Object, usually for the purpose of performing I/O operations or configuring the I/O service.
Service Object	An addressable entity in an I/O Unit, such as an I/O Service Object or I/O Management Object, that provides some type of service to a Service

Client. Thus, it is a port through which a client can access Protocol Objects or manage I/O devices. The characteristics of a Service Object is that it consumes at least one QP and it does not share its QPs with other Service Objects. That is, Service Objects are addressed via their QPs.

Supervisor

The entity on a Client Platform (a.k.a. I/O Resource Manager or IORM) that manages I/O clients on that platform.

Supervisor_Key

A key a supervisor (i.e., the IORM of a Client Platform) passes to an I/O Unit that the I/O unit compares to keys in the I/O unit's Platform Pool Table. The I/O unit allows the IORM access to DevMgt attributes based on the record that matches this key. IORMs use the key to read its Platform Pool Table record and to modify Client Pool Table records associated with that Supervisor_Key to further control which I/O unit resources each of its clients may access.

A7.1.2 COMPLIANCE

This annex specifies compliance requirements for a configuration manager (Device Manager and its Device Administrator) and for client platforms.

This annex specifies two new Compliance Categories (see Chapter 20: Volume 1 Compliance Summary on page 1072 for explanation of compliance categories and qualifiers). The new categories are:

- Configuration Manager - the application that provides the Device Manager and the Device Administrator for a configuration group)
- Client Platform - a platform that uses an I/O units's resources.

There are two Compliance Qualifiers for the Configuration Manager category and no Compliance Qualifiers for the Client Platform category. The compliance categories for Configuration Managers are:

- PERS - The manager persistently saves client contexts such as subscriptions and diagnostic sessions across reset, restarts, and power cycles.
- FAILOVER - The manager supports graceful failover such that if the manager fails and a standby manager takes over, the new manager inherits the client context (subscriptions and diagnostic sessions) from the old manager.

Section A7.7 Compliance on page 1503 provides a summary of compliance statements.

A7.2 OVERVIEW

When an IOU can be accessed by multiple client platforms, there is a need for a management entity (i.e., a Configuration Manager) that admin-

isters assignment of service objects to the various client platforms and coordinates actions that might impact the client platforms, such as invoking diagnostics and hot plugging an I/O module.

Assignment of IOUs to client platforms is not a one to one relationship and needs to take into consideration sharing aspects. An IOU is considered shared when more than one client platform has access to that IOU's resources.

[Editorial Note:For the simple case, each client platform is a host platform running an operating system that supports multiple clients which initiate I/O transactions, but in elaborate I/O architectures, an IOU could also be an initiator of I/O requests to another IOU, and therefore, for the purpose of managing the relationships between those IOUs, the initiating IOU is considered a client platform. Thus, Device Management / Device Administration controls the client platform / IOU relationship and any node that needs access to an IOU's service object is considered a client platform.

To enable graceful sharing of an IOU between independent client platforms, a Configuration Manager becomes necessary and it:

a) Acts as the central class manager for DevMgt agents of IOUs in the configuration manager's configuration group.

b) Provides each client platform with information necessary for that platform to locate and use I/O services by passing out lists of IOUs and access keys to each client platform

c) Provides a central service for maintaining I/O assignment information and:

   iii) Provides centralized control for approving a Diagnostic Session.

   iv) Provides I/O module Hot Plug/Removal notification to affected clients

   v) Performs I/O resource polling functions that would otherwise be replicated by each client platform

The Configuration Manager in conjunction with the Device Management Agent provides:

- Individually assigning each service object of an IOU to one or more client platforms.
- Notifying client platforms about configuration changes and configuration events.
- Mechanism for graceful insertion and removal of I/O-Modules (hot plug and hot swap).

- Coordination of diagnostics.
- An efficient scalable means for client platforms to discover applicable IOUs and to query for configuration information.
- The means for client platforms to discover IOUs on other subnets.

A7.2.1 OBJECTIVE

The goals of the configuration management framework are:

- Provide the means for a Configuration Manager to allocate I/O resources by configuring each IOU, specifying which service objects and associated resources that each client platform may use.
- Allow each client platform's OS the ability to allocate those resources to clients within that platform, specifying which subset of those service objects and resources each client may use.
- Enforce allocations such that client platforms and their clients cannot consume resources for which they are not authorized.
- Coordinate events that affect multiple client platforms.

A7.2.2 USAGE MODEL

The Configuration Manager is an InfiniBand management facility that manages relationships between client platforms and I/O resources within a managed IOU. It is composed of a Device Manager (DM) and a Device Administrator (DA), which are described as two separate entities as a matter of architectural convenience (see Figure 293 on page 1443).

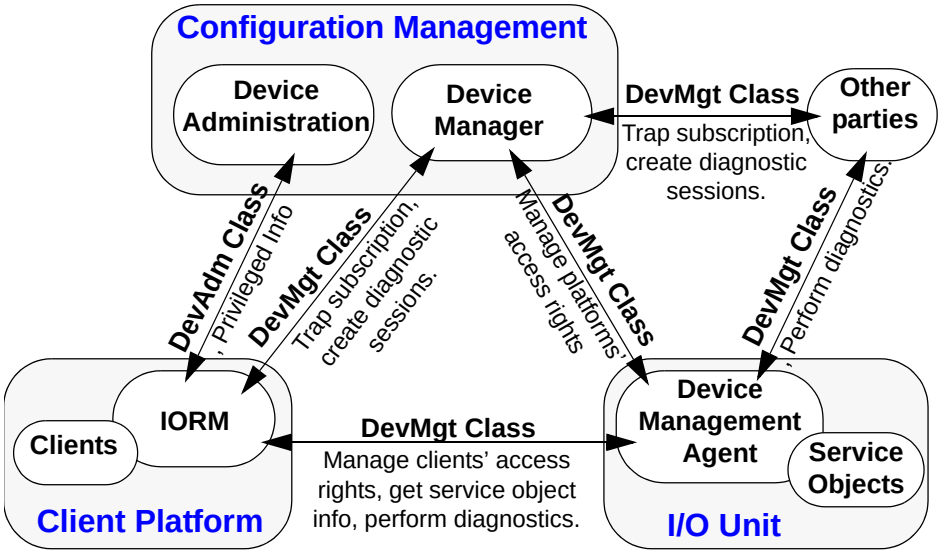


Figure 293 Configuration Management Usage Model

The DA is the complement to the DM. The DM uses DevMgt class MADs[42] to configure each IOU with information describing which platforms may access which service objects. It does this by setting Platform Pool Table records in each IOU (one record per client platform), identified by a unique supervisor key). Each Platform Pool Table record specifies which service objects and I/O resources the platform may use.

.

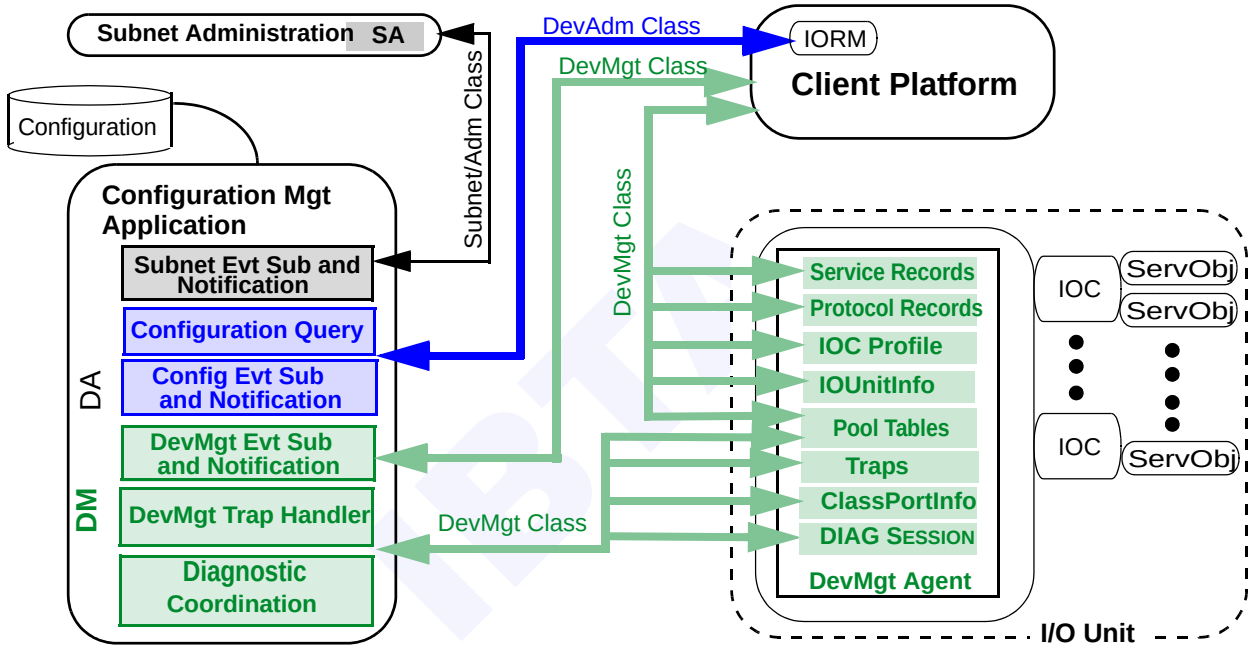


Figure 294 Configuration Management Functions and Relationships

Before a client platform can query the DA, the platform's I/O resource Manager (IORM) must first login to the DA so that the DA can validate the client platform and assign it a RequesterID. The IORM uses that RequesterID in subsequent DevAdm MADs.

The IORM may then query the DA for information about IOUs it is permitted to use using DevAdm MADs. The IORM is a privileged process on a client platform that uses a special privileged Q_Key to query the DA for privileged information about I/O resources for that platform. Specifically, the DA provides the IORM with a list of IOUs and Supervisor_Keys. The Supervisor_Key authorizes the IORM to access the IOU and read that platform's Platform Pool Table record (to learn which resources it may use) and to configure IOU Client Pool Table records assigned to that plat-

42. The DevMgt Annex describes all of the DevMgt attributes including the communication between clients and the Device Manager for trap subscriptions and forwarding (requirements for trap forwarding are specified in this annex).

form. Each Client Pool Table record is identified by a Client_Key and specifies which of the resources assigned to the platform that a client using that Client_Key is allowed to use.

The IORM then distributes client keys to its clients as necessary. Each client uses its client key to access the IOU using DevMgt MADs to read information about service objects associated with that client key. The client also uses that key when creating a connection with the service object. Refer to the Device Management Annex for details on how the IORM uses its supervisor key and provisions an IOU with client access privileges.

It is important that only the platform's IORM, and not the platform's clients, have access to the platform's supervisor key. Thus, DevAdm uses a special privileged Q_Key. Since the OS controls who can use privileged Q_Keys, the OS has the means to prevent its clients from querying the DA to learn privileged information. In addition, the DA validates the identity of the client platform before providing that platform with any configuration information (see A7.6.3.4 "LogIn" on page 1491) and only provides information for that platform.

The DA also coordinates events such as I/O module hot-plug/removal and diagnostics with client platforms that are assigned service objects affected by those events. The DA notifies affected clients via Event Subscription and Notification, and the response from the clients determine whether the DM continues with the hot plug or diagnostic session.

In addition, the DM provides an interface (using DevMgt MADs) that is used by both client platforms and other managers to subscribe to DevMgt class traps and to create diagnostic sessions. Thus, client platforms and other managers use DevMgt class to access the DM function of the Configuration Manager.

A7.2.3 CONFIGURATION MANAGEMENT APPLICATION

A configuration manager consists of a DM and its associated DA. The Device Management class provides mechanisms (methods and attributes) for a DM to configure IOUs and the means for client platforms to query the IOU about those service objects that the DM had configured for that platform (see Annex A8: Device Management).

The Device Administration class provides mechanisms (methods and attributes) to enable an IORM to query the DA to retrieve supervisor keys and for the DA to report configuration changes to the IORM.

A7.2.3.1 PASSIVE MANAGEMENT

It is not always necessary to have a Configuration Manager (i.e., a DM and DA). The term "*Passive Configuration Management*" refers to a fabric

designed to function without a configuration manager. Passive management is typical in a subnet where each IOU is dedicated to a single client platform (example: each IOU configured for a single partition containing a single client platform) or where all clients in that partition have equal access to all of the IOU's resources. Except for the most trusted environments, a Configuration Manager is needed when IOUs are shared among multiple clients. For passive configuration management, DevMgt class defaults (i.e., KeyInfo:ProtectBits) allow any client platform that can access an IOU (i.e., has the proper P_Key) to access to all of the IOU's Device Management information and I/O resources.

A7.2.3.2 ACTIVE MANAGEMENT

For active device management there is a configuration manager that provides a DM and a DA.

In general, a DA:

- Provides each IORM with a list of its IOUs and supervisor keys.
- Notifies client platforms when IOUs, IOCs, or service objects come on-line or go off-line.
- Notifies client platforms about changes in configuration.
- Manages hot plug events - notifying client platforms when one or more I/O Controllers are to be removed.
- Notifies client platforms about diagnostic sessions.

And Client Platforms and other managers use the DM to:

- Register for and receive reports of IOU generated DevMgt Traps.
- Request a diagnostic session with an IOU.

A7.2.3.3 MULTIPLE MANAGERS

Configuration Management supports the concept of multiple configuration managers for the following reasons.

- Segregated resources - when a fabric is divided into two or more independent configuration groups, each group will have its own logical Configuration Manager, independent of the other groups as illustrated in Figure 295 on page 1447.
- Redundancy - standby Configuration Managers monitor active managers, ready to takeover in case of failure as illustrated in Figure 296 on page 1448. Only the active DM registers with the SA, so a client platform does not see standby Configuration Managers.
- Distributed management - for large fabrics, having multiple Device Managers help balance the workload. In this case, the discrete managers cooperate to form a single logical Device Manager. This means that two clients of the same configuration group might not see the same DM / DA as illustrated in Figure 297 on page 1448.

A client platform (IORM, I/O client, I/O management application) might see multiple DMs (one for each partition) if it belongs to multiple groups (e.g., 'Client Platform n' in Figure 295). Since a DM might register for multiple partitions, it is possible for a client platform that only belongs to a single configuration group to get multiple ServiceRecords for the same DM when it queries the SA to locate the DM. When a client platform does get multiple ServiceRecords, it can query each DM for the DAInfo attribute and the ConfigGroupID in the DAInfo Attributes will indicate which DMs serve different configuration groups.

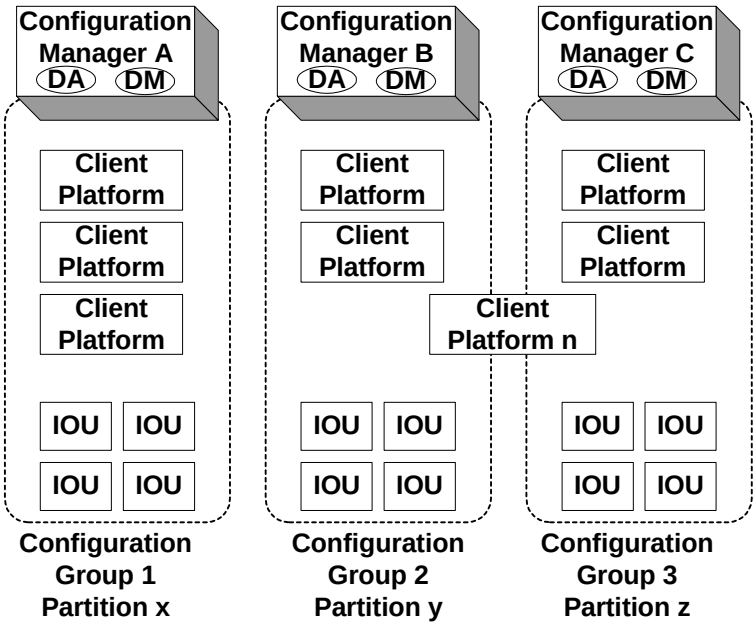


Figure 295 Multiple Configuration Group Example

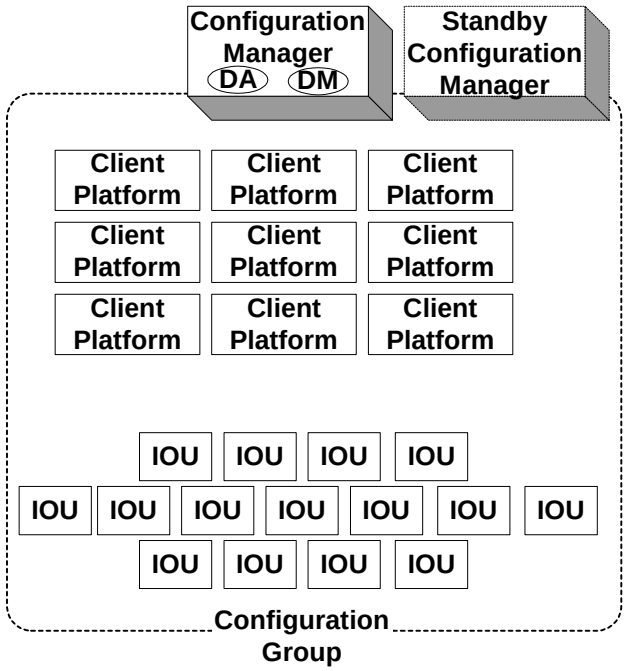


Figure 296 Standby Configuration Manager Example

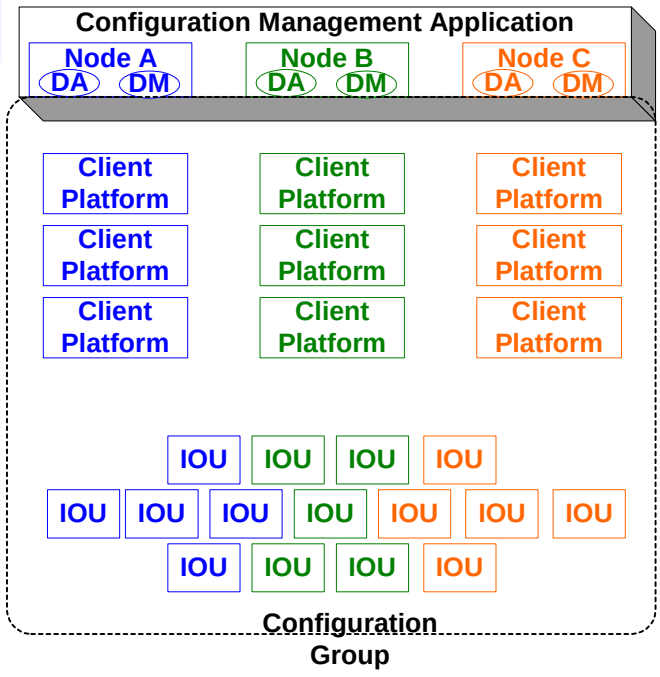


Figure 297 Distributed Configuration Manager

Unlike the SM/SA, there does not need to be a DM/DA per subnet. That is, one configuration manager can handle multiple subnets. However, it might be desirable to provide a configuration manager per subnet. For example, each DM/DA in Figure 297 might serve a different subnet. In this case, each DM would register [via SubnAdmSet(ServiceRegistration)] on its own subnet and directly manage the IOUs on its own subnet, with the understanding that all the DM/DAs coordinate privately and appear as a single configuration manager. Thus, a client platform queries its DA to locate all of its I/O resources, even if those IOUs are on a different subnet.

A7.3 CONFIGURATION MANAGEMENT OPERATIONAL MODEL

The Configuration Manager is a management facility that appears as a service (i.e., the DA) to client platforms and as a class manager (i.e., the DM) for DevMgt Agents as illustrated in Figure 298 on page 1449. Each client platform is able to access IOUs directly using DevMgt class MADs. However, it is the Configuration Manager (i.e., the DM) that has the responsibility and the authority (by way of the Manager_Key) to initially configure the IOU and assign resources to client platforms. And it is the DA that provides client platforms with the keys that they need to manage the resources assigned to the client platform.

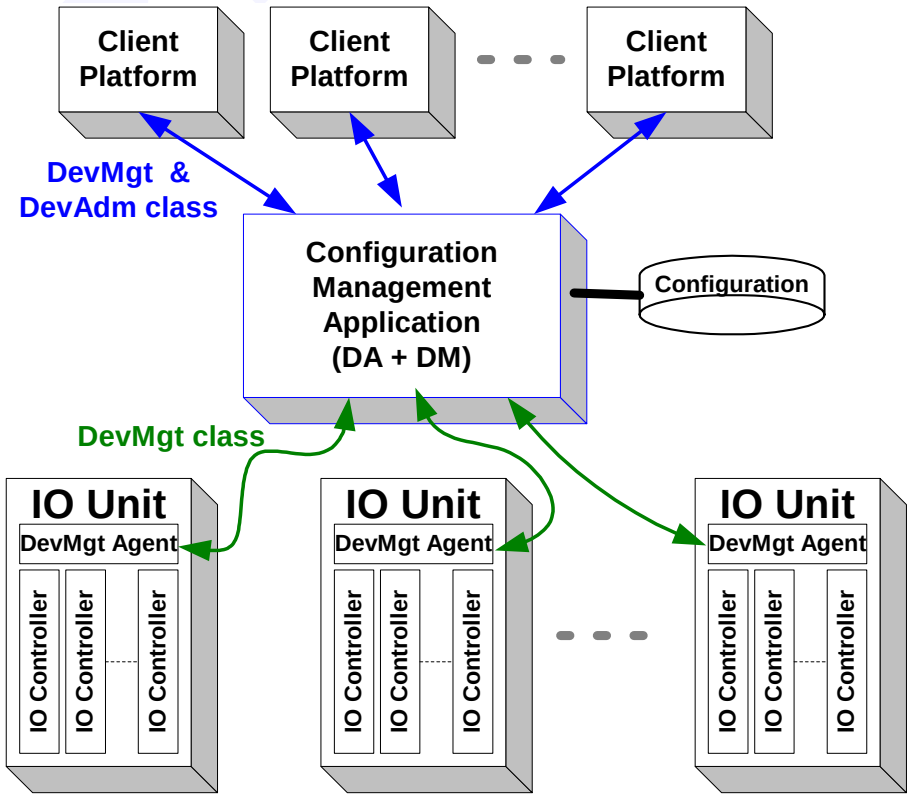


Figure 298 Configuration Management Operational

Configuration management provides the means to manage the sharing of service objects and other IOU resources by client platforms. The Configuration Manager performs common maintenance functions to build and maintain its configuration information, thus providing client platforms with a single interface (the DA) for acquiring information about IOUs. Each platform's IORM has privileged access to that platform's configuration information and supervises that platform's I/O operation by directly configuring client pools on each IOU and providing its clients with the information they need to directly access the IOU's services.

A Configuration Manager reduces resource management complexity by providing client platforms with a central facility that handles event notification and coordination as follows.

- The DM provides subscription service that allows client platforms to subscribe for DevMgt Traps.
- The DM provides the interface for requesting a diagnostic session with an IOU.
- The DA provides a notification service that informs client platforms about configuration changes, I/O module hot plug/removal events, and diagnostic sessions.
- Together, the DM & DA provide coordination for I/O management actions such as I/O module hot plug/removal and system configuration change.
- The Configuration Manager reduces management traffic by providing a central agent that performs polling functions that otherwise would have to be performed by each client platform. For example, the Configuration Manager detects when IOUs or IOCs come on-line and report those events to client platforms.

A7.4 CONFIGURATION MANAGEMENT CHARACTERISTICS

A7.4.1 CONFIGURATION DOMAIN

1) The term '**configuration group**' denotes the set of IOUs assigned to a Configuration Manager. Thus, by definition a Configuration Manager manages exactly one configuration group. This term is used for describing the scope of responsibility for a Configuration Manager.
2) An IOU belongs to only one configuration group, the one that sets the IOU's Manager_Key. This is necessary because IOU traps can only be sent to one destination (i.e., one DM). For the case where an IOU has multiple ports, each port is capable of sending traps to a different DM. However, those DMs need to cooperate in some fashion for information to be accurate. Cooperating DMs are, for all practical purposes, one logical DM.
3) Each configuration group is identified by a ConfigGroupID.

4) A Client Platform may have access to multiple configuration groups as permitted by the client platform's P_Key table. That is, it could see multiple Device Managers, each managing a different configuration group. It might also see multiple managers serving the same configuration group.

5) If an IOU is not in the Configuration Manager's configuration group, the default policy is that the Configuration Manager does not automatically acquire it, since that IOU might already be a member of another configuration group. How an IOU gets added to a Configuration Manager's configuration group is outside the scope of this annex.

6) Logically, there is only one DM and DA for a particular configuration group. There may be standby managers and/or the functionality might be distributed. However, client platforms are not aware of standby managers.

A7.4.2 PARTITION USAGE

7) The proper partitioning must be in place which allows the IORM to communicate with the Configuration Manager, the Configuration Manager to communicate with the IOU, and the client to communicate with the IOU.

8) For the purpose of explaining the architecture, an '**I/O partition**' is defined as the set of client platforms and IOUs that share the same P_Key value for the purpose of supporting I/O operation. Either the client platforms or the IOUs (or both) have full membership capability as defined by the most significant bit of their assigned P_Key value. Thus, an I/O partition exists by virtue of the SM assigning the same P_Key value to an IOU and a host.

9) A single Configuration Manager may manage I/O resources for multiple I/O partitions and the Configuration Manager does not need to be a member of each I/O partition. For example, there could be one partition where the Configuration Manager has full membership and each IOU has limited membership as determined by the most significant bit of the P_Key. The purpose of this partition is communication between the DM and the DevMgt Agent and not for I/O. Of course, IOUs that support Device Management should provide multiple entries in its P_Key tables (one for the Configuration Manager and one for each I/O partition). Otherwise, the Configuration Manager would need to be a member of each I/O partition.

10) The partition that a client platform uses to access the DA does not need to be a partition that the platform uses to access IOUs. For example, there could be one partition where the DA has full membership and each client platform has limited membership as determined by the most significant bit of the P_Key. This could be the same partition that the DM uses to communicate with DevMgt Agents.

11) There can be multiple paths because of partitioning. When more than one I/O partition exists between a client platform and an IOU, selection resides with the client platform, and may be governed by path information provided by the SA.

12) The Configuration Manager obeys partition rules. This does not prevent the Device Manager from forwarding information such as notices using a different partition, but the Configuration Manager does verify that a subscribing platform is permitted access as per Chapter 13 "*Management Model*" section 13.4 "*Management Datagrams*" subsection titled "*Event Forwarding*".

A7.4.3 SA USAGE

13) The DM registers with the SA so that client platforms and other managers can locate the Configuration Manager.

14) The DA uses SA services to detect IOUs that come on-line so it can report those events to client platforms. It also monitors traps from IOUs to detect IOCs that come on-line so it can report those events to client platforms.

A7.4.4 MANAGER INTERACTION

15) How a Configuration Manager stores its data and the communication between an active and a standby manager is outside the scope of this specification. It is expected that a single vendor provides both the active and standby managers for a particular configuration group.

16) This annex does not specify the protocol between standby and active managers, nor does it specify how a master DM is elected. It is expected that a single vendor provides both the active and standby managers for a particular configuration group.

17) The Configuration Management framework does provide for multiple independent Configuration Managers, each serving a different configuration group. Communication between Configuration Managers that serve different configuration groups is not necessary.

A7.5 CONFIGURATION MANAGEMENT OPERATION

Configuration management is an IB management application that an administrator uses to manage which I/O resources may be used by which client platforms as illustrated in Figure 294 "Configuration Management Functions and Relationships" on page 1444.

The Configuration Manager interacts with the SubnAdm (SA) agent of the SM, with DevMgt class agents of IOUs under its control, and with the client platforms which it serves.

Typically, a Configuration Manager uses SA services to detect IOUs coming on-line and uses IOU traps to discover IOCs coming on-line, notifying client platforms appropriately.

This section specifies the overall operation for the Configuration Manager. The following section (A7.6 "DevAdm Class Definition" on page 1471) specifies the wire level protocol for the DA portion of the Configuration Manager and Annex A8: Device Management specifies the wire level protocol for the DM portion of the Configuration Manager.

A7.5.1 INTERACTION WITH SUBNET MANAGER

Configuration Management complements subnet management, but the Configuration Manager is not necessarily collocated with the SM or SA. In fact, there may be multiple Configuration Managers per subnet and a single Configuration Manager can span several subnets.

The I/O partitions, partitions used for client platform to DA communications, and partitions used for DM to IOU communications are subject to subnet management policy. For example, there might be one partition that all client platforms use to access the DM / DA, or perhaps a different partition per client platform. One use of partitioning would be a single partition dedicated for configuration management where the DM / DA has full membership (as per the P_Key msb) and all client platforms and IOUs are limited members and thus can only communicate with the DM / DA (not each other). Additionally, assignment of I/O resources to client platforms requires appropriate I/O partitions.

Thus, Configuration Management is dependant on partitioning created by the SM. How a Configuration Manager coordinates partition usage with the SM is outside the scope of this document.

A7.5.2 INITIALIZATION AND SA REGISTRATION

The configuration management framework supports multiple Configuration Managers per subnet. Each Configuration Manager registers with the SA so that client platforms and interested parties can locate it.

The DM function of the Configuration Manager registers with the SA via the SubnAdmSet(ServiceRecord) as specified in 15.2.5.14 ServiceRecord on page 895, to advertises it's location. Client Platforms and Interested parties that choose to subscribe to DevMgtTraps can find DMs by querying the SA. The client can then query the DM to get the ConfigGroupID and the address of the DA.

The DM uses the well-known management Q_Key 8001_0000. Thus, any agent on any node allowed to use that Q_Key can subscribe with the DM for DevMgt traps. Once a client platform locates the DM, it can send a DevMgtGet(DAInfo) to the DM to get the address (LID/GID/QPN) of the DA. The DA uses the special Q_Key 8001_0001, which is reserved for DA to IORM communications. The OS should only permit the IORM to use this Q_Key, so that only the IORM can communicate with the DA on behalf of that platform.

CA7-1: A DA **shall not** respond to DevAdm MADs that do not use the special DevAdm Q_Key.

If redundant managers exist, they are expected to be provided by the same vendor. Thus, the protocol that redundant managers use to validate that they serve the same configuration group, how they elect a master, and how they coordinate and maintain consistency is outside the scope of this document.

CA7-2: A Device Manager **shall** register its services with the SA via SubnAdmSet(ServiceRecord) using the ServiceName “DeviceManager.IBTA” prior to sending a DevMgtSet(KeyInfo) to a DevMgt agent. See 15.2.5.14 ServiceRecord on page 895 and Service Names in Annex A3 “Application Specific Identifiers”. If the Device Manager is configured for multiple partitions, it shall register once for each partition.

If the SA rejects a Device Manager’s registration, it means that the Device Manager is not authorized to operate on that subnet.

CA7-3: If the SA rejects a Device Manager’s SubnAdmSet(ServiceRecord), the Device Manager shall not send DevMgtSet(KeyInfo) to any IOUs on that subnet.

The Device Manager is responsible for periodically renewing its lease with the SA to prevent the SA from dropping its Service Records (see15.2.5.14.3 ServiceLease on page 898).

CA7-4: The Device Manager shall renew its registration lease by reregistering with the SA before its service lease expires.

When the configuration management application is distributed over multiple nodes (i.e., distributed management), only one node registers as a Device Manager per partition per subnet. That node can distribute the workload by using MAD redirection if desired.

A standby Device Manager that transitions to active should register itself with the SA and remove the old registration.

A7.5.3 COHERENCY BETWEEN CONFIGURATION MANAGERS

A Configuration Manager establishes DevMgt ownership of an IOU by setting the IOU’s Manager_Key via a DevMgtSet(KeyInfo). If the Configuration Manager is unable to set the Manager_Key it means that another manager has ownership. There are a number of reasons for such a situation and this annex does not address how to remedy this situation other than suggesting that the Configuration Manager report the conflict through a user or management interface. Such an interface is outside the scope of this specification.

CA7-5: When two DMs have the same ConfigGroupID, they shall work in unison and provide the same information and service to a client platform as if they were two different ports of the same DM.

CA7-6: If the DM is not able to establish control of an IOU by setting the IOU's Manager_Key, then the Configuration Manager shall not include the IOU or any information regarding the IOU in any communication with client platforms.

CA7-7: Should a DM be elected master DM, its DA shall also be implicitly elected master. If a DM ceases to be master, its DA shall cease to be master.

CA7-8: Should a DM cease being the Master DM (e.g., transitions from master DM to standby DM), it shall reject DevMgtGet() requests using MAD Header Status [8:15] = 'NotMasterDM' and cease sending DevMgtTrapRepress() and DevMgtReport() messages.

CA7-9: Should a DA cease being the Master DA (e.g., transitions from master DA to standby DA), it shall reject DevAdmGet() requests using MAD Header Status [8:11] = 'Not Master DA' and cease sending DevAdmReports()s.

For distributed management, multiple Device Managers may attempt to access the same IOU. Access to that IOU should be serialized to prevent race conditions and integrity issues that result when multiple Device Managers are reading and writing attributes at the same time. One way to achieve atomic access is by the Device Manager temporarily changing the KeyInfo:Manager_Key before it performs any other DevMgtSet()s and then change it back when it is done. Thus, if a Device Manager is not able to change the key, it means that another manager performing an update.

A7.5.4 ACTIVE VS. PASSIVE CONFIGURATION MANAGEMENT

For environments that do not contain a Configuration Manager (i.e., no DM / DA), a client platform needs to be able to discover and use IOUs (passive management). When a DM /DA is present (active management), a client platform cannot arbitrarily use IOUs. If the client platform determines that there is no DM, then it can use IOUs that have the IOUnitInfo:IsActivelyManaged bit set to zero. This bit is automatically set when the DM sets the IOU's Manager_Key to a non-zero value.

CA7-10: A Client Platform shall not access a service object unless either a DM / DA has assigned that service object to the client platform or the IOU's IOUnitInfo:IsActivelyManaged bit is zero.

Only the SM and DM should have access to an IOU prior to the DM setting the IOU's Manager_Key and configuring the IOU's pool tables. Otherwise,

there could be a potential race condition where a client platform emerges before the Configuration Manager, incorrectly determines that an IOU is not actively managed, and thus attempts to use IOUs. In an ideal subnet, the SM does not configure an IOU with I/O partitions until after the DM configures the IOU. This prevents a client platform from seeing un-managed IOUs, and thus prevents conflicts.

Because coordination between SM and the Configuration Manager is not guaranteed, a Client Platform should wait for a sufficient time interval before using an IOU. For passive management, it is desirable that the interval be extremely small and for active management, it is desirable the interval be extremely large. Thus, such a parameter is subject to local policy. It is recommended that a client platform persistently remember if a DM / DA previously existed and set its time-out accordingly.

A system administrator should be able to add new equipment to a subnet with minimal risk and without requiring the system administrator to configure the equipment before installing it. Thus, the factory default timeout for determining passive management mode should be long.

CA7-11: The factory default time that a client platform waits before it determines passive management and starts to use IOUs shall be no less than 2 minutes.

A7.5.5 DEVICE MANAGER OPERATION

The DM uses DevMgt class MADs (methods and attributes) to set and retrieve IOU DevMgt information. The DM communicates with the DevMgt agent by sending a DevMgtSet() or a DevMgtGet() to the DevMgt agent and the DevMgt agent responds with a DevMgtGetResp(). The DevMgt agent also sends DevMgtTrap()s to the Device Manager repeating each trap until the Device Manager responds with a DevMgtTrapRepress().

CA7-12: DevMgt class datagrams **shall** conform to the MAD format and use as specified in 13.4 Management Datagrams on page 717 and further specified in Annex A8: Device Management Figure 316 Device Management MAD Format on page 1530 and Table 447 Device Management MAD Fields on page 1531.

CA7-13: The datagrams for the DevMgt and DevAdm class **shall** conform to the Common MAD requirements as specified in 20.14 “Common Mad Requirements”.

CA7-14: A DM shall respond to MADHeader:ClassVersion values as per Annex A8: Device Management Table 448 "Class Version" on page 1532

CA7-15: The DM shall not check the MADHeader:Access_Key in any received DevMgt class MAD.

One way for a Configuration Manager to determine if a DevMgt agent resides on an IOU is to query the SA using SubnAdminGet(PortInfoRecord) for any port on the IOU. By inspecting the CapabilityMask:IsDeviceManagementSupported in the PortInfoRecord query returned by the SA, the Configuration Manager determines if the DevMgt agent exists. If so, the Device Manager can determine if the DevMgt agent supports Traps or Notices by testing if DevMgtGetResp(ClassPortInfo:CapabilityMask(0)) = 1b and DevMgtGetResp(ClassPortInfo:CapabilityMask(1)) = 1b respectively. Note that version 2 of Device Management requires DevMgt Agents to support Traps. Traps allow the DevMgt agent to notify the Device Manager of events such as security violations and changes to the IOU's I/O capability.

Third parties can register to receive DevMgt Management Traps by issuing a DevMgtSet(InformInfo) to the Device Manager (see 13.4.3.8 "Informinfo"). When a trap event occurs on the IOU, the IOU sends a DevMgtTrap(Notice) to the Device Manager. The Device Manager forwards the Trap to the third party through the DevMgtReport(Notice). For more information on DevMgt Management Traps see the DevMgt Annex.

A Device Manager supervises diagnostic sessions. Clients and other management programs have to request a diagnostic session by sending a DevMgtSet(DiagSession) to the DM. If the DM approves, the DA notifies the affected clients and then DM sets up the diagnostic session with the IOU by sending a DevMgtSet(DiagSession) to the IOU. The diagnostic initiator invokes the diagnostics by sending DevMgt MADs directly to the IOU. When the diagnostic initiator finishes testing, it terminates the session by sending a DevMgtSet(DiagSession) to the DM, which terminates the session by sending a DevMgtSet(DiagSession) to the IOU and then the DA notifies client platforms that the testing has ended via IOC On-Line reports (see A7.5.9 Diagnostics on page 1468 for details).

A7.5.6 PROTECTING THE MANAGER_KEY

By definition, when a node sets the Manager_Key in an IOU, it becomes the DM. Once the node becomes the DM controlling an IOU it has the responsibility to periodically access the IOU so that the IOU knows that the DM is active and has not failed. The IOU's Manager_Key lease period is the mechanism to allow another DM to take control in case the current DM fails.

A Device Manager sets the IOU's KeyInfo:ProtectBits to protect against others learning the IOU's KeyInfo:Manager_Key. However, these bits are subject to being reset if the Manager_Key lease period expires, which could allow unauthorized access. A DevMgt agent resets its lease period timer when it receives a MAD with KeyType=DM and a valid MADHeader:Access_Key. Thus, a Device Manager can protect against unauthorized access by sending each of its DevMgt agents a DevMgt MAD

with a valid MADHeader:Access_Key before the DevMgt agent's Manager_Key lease period expires.

CA7-16: When a Device Manager sets a non-zero KeyInfo:Manager_Key and non-zero KeyInfo:ProtectBits in an IOU, the Device Manager shall continuously reset the lease period counter by sending a DevMgt MAD with MADHeader:Access_Key equal to the IOU's KeyInfo:Manager_Key at an interval less than the lease period.

A7.5.7 I/O UNIT TRAP FORWARDING

Device Management uses the event reporting framework defined in chapter 13 (see 13.4.9 Traps, 13.4.10 Notice Queue, and 13.4.11 Event Forwarding).

A7.5.7.1 CONFIGURING IOUS FOR TRAPS

The DM configures each IOU in its configuration group so that the IOU sends DevMgt class traps to the DM. It does this by setting the TrapGID, TrapTC, Trap FL, TrapHL, TrapLID, TrapQP, TrapQ_Key, TrapSL, and TrapP_Key components in the DevMgtSet(ClassPortInfo) of at least one port of each IOU.

CA7-17: The Device Manager shall set the ClassPortInfo Trap components of the DevMgt Agent for each IOU in its configuration group such that the DevMgt agent sends DevMgtTrap()s to the Device Manager.

CA7-18: When a Device Manager receives DevMgtTrap(), it shall respond with a DevMgtTrapRepress().

A7.5.7.2 TRAP SUBSCRIPTION / REPORTING

A node may subscribe for DevMgt Trap forwarding by issuing a DevMgtSet(InformInfo) to the DM. The LID Range and GID in the InformInfo attribute indicates for which IOU the client is interested in receiving Report()s. Note that when subscribing and unsubscribing for the Heartbeat, the GID and LID Range components in the InformInfo is irrelevant and thus the client should set GID, LIDRangeBegin, LIDRangeEnd to zero, 0xFFFF, & zero respectively and the DA shall ignore those components.

When the DM receives a DevMgtTrap(), it forwards the trap to subscribed parties via the DevMgtReport(Notice) with the exception of traps listed in Table 423: Privileged Traps. Traps in Table 423 are considered private and must not be forwarded except as indicated in the table.

Table 423 Privileged Traps

Trap	Trap
DiagSessionState	This trap contains the DiagToken and is considered private. The DM shall only send the DevMgtReport() to the node that initiated the Diag Session.
MgrKeyViolation, SupvKeyViolation, Client Violation, DiagToken Violation, DiagSession Violation	These security traps contain information which might be considered sensitive (such as invalid keys and DiagTokens). The DM may restrict which nodes receive the DevMgtReport() and/or may zero the key/DiagToken component in the Notice attribute. The DM policy can be on a trap by trap basis. If a node attempts to subscribe to one of these traps individually and the DM policy is not to forward the trap to that node, then the DM rejects the DevMgtSet(InformInfo) with MAD Status [8:15] = 'PolicyReject'. However, the DM accepts a subscription for 'TrapNumber=0xFFFF' regardless of whether it forwards these traps.

CA7-19: When a Device Manager receives a DevMgtTrap() not listed in Table 423 "Privileged Traps", it shall generate a DevMgtReport() to all parties that have subscribed with the Device Manager for that notice if a GID of the IOU matches the InformInfo:GID or the IOU has a LID that falls in the range of the InformInfo: LIDRangeBegin – LIDRangeEnd.

CA7-20: When a Device Manager receives a DevMgtTrap() listed in Table 423 "Privileged Traps", it shall only generate a DevMgtReport() as described in the table.

CA7-20.2.1: If the Device Manager rejects a DevMgtSet(InformInfo) because it does not permit subscription to that trap, it shall reject the request using MAD Status [8:15] = 'PolicyReject'.

Because DevMgt Agents are required to support Traps, the Device Manager is not required to poll notice queues, and does not generate DevMgtReport()s for Notices it reads from a notice queue.

CA7-21: The Device Manager shall not generate DevMgtReport()s for Notices it reads from a notice queue.

The sequence of message exchanges for trap subscription mechanism is shown in Figure 299 and Figure 300.

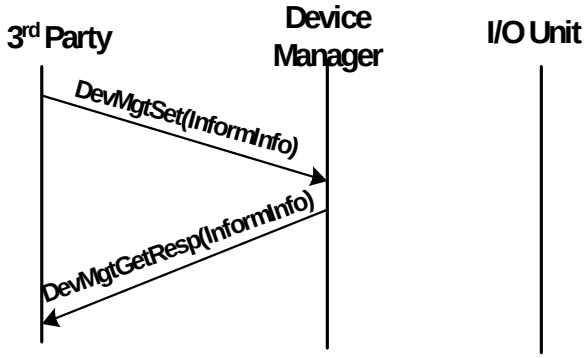


Figure 299 Trap Subscription

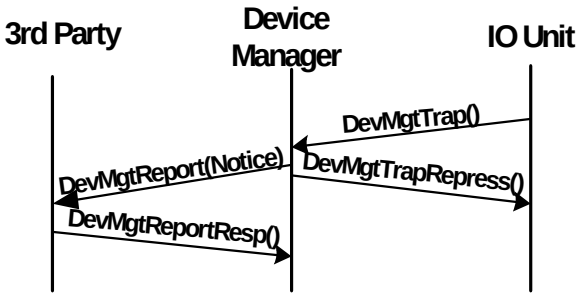


Figure 300 Trap Forwarding

The manager reports traps in the order that it receives them from the IOU. When the manager receives multiple traps from the same IOU, it delivers them one at a time in the order that the traps were received and waits for a ReportResp() from the client before delivering the next Report().

CA7-22: A DM shall report Traps from an IOU in the order it receives the traps. However, the DM is not required to maintain ordering of traps from different IOUs.

CA7-23: A DM shall only have one DevMgtReport() outstanding per subscriber per IOU. That is, the DM shall wait for a DevMgtReportResp() before sending a subsequent DevMgtReport() for another trap from the same IOU.

[Editorial Note:A Class Version 2 DM does not forward Class Version 1 Traps. If a client desires to subscribe for Class Version 1 Traps, it can issue a Class Version 1 DevMgtSet(InformInfo) to the DM. If the DM does not support Class Version 1, it rejects the request with 'MAD Status 2:4 = 001 - Bad version' (version not supported).]

A7.5.7.3 SUBSCRIPTION INTEGRITY

Clients depend on receiving Report()s whenever an IOU generates a trap and thus depend on the DM reporting traps to subscribers. However, there are a number of events that can cause a subscription to be destroyed where the client might not be aware that its subscriptions were destroyed.

- DM reset - In the event that the DM is reset, subscriptions can be lost. It is recommended that the DM retain subscription information in persistent storage, such that subscriptions survive power cycles and DM resets. A client can subscribe with the SA for Trap 64/65 ‘Port In/Out of Service’ to detect when the node on which the DM resides is reset. The client can also subscribe with the DM for the Heartbeat notice (see A7.5.7.5 "Heartbeat" on page 1463), so it can detect when the DM disappears or ceases to function. Successful reception of the heartbeat indicates that the DM is alive and that the client’s subscriptions are still valid.
- DM failover - In the event that the active DM fails and another DM takes over, subscriptions with the old DM might not be carried over to the new DM. For graceful failover, the active DM should provide subscription information to standby DMs before replying to a subscription request. A client can subscribe with the SA for Trap 65 ‘Port Out of Service’ to detect when the node on which the DM resides goes down and subsequently query the SA to locate the new DM. If the client subscribed with the DM for the Heartbeat notice (see A7.5.7.5 "Heartbeat" on page 1463), the heartbeat can indicate graceful failover. Successful reception of the heartbeat indicates that the new DM inherited the client’s subscriptions.
- Temporary path disruptions can make a client unreachable, which might result in the DM timing-out and removing the client’s subscriptions. Both the client and the manager can subscribe to SA trap 65 to detect such a condition (assuming that the SM detects the path failure). Again, the client can subscribe with the DM for the Heartbeat notice (see A7.5.7.5 "Heartbeat" on page 1463), so it can detect when the DM drops its subscriptions. Successful reception of the heartbeat indicates that the client’s subscriptions are still valid.

oA7-1: If the Device Manager supports Persistent Context (i.e. sets the *IsContextPersistent* bit in DevMgtGetResp(ClassPortInfo) to 1), it shall retain subscriptions across power cycles (i.e., use subscription information stored in non-volatile storage) and shall also retain information about current Diag Sessions.

oA7-2: If the DM supports Persistent Context (i.e. sets the *IsContextPersistent* bit in DevMgtGetResp(ClassPortInfo) to 1), it shall save subscrip-

tion information in non-volatile storage before responding to the DevAdmSet(InformInfo).

oA7-3: If the Device Manager supports Graceful Failover (i.e., sets the *GracefulFailover* bit in DevAdmGetResp(ClassPortInfo) to 1), it shall share subscription information with all standby Device Managers for that same configuration group provided by that same vendor. This includes supplying Standby DMs with subscription information when the standby manager comes on-line and updating standby DMs when subscription information changes. It also includes sharing Diag Session information.

oA7-4: If the DM supports Graceful Failover (i.e., sets the *Graceful-Failover* bit in DevMgtGetResp(ClassPortInfo) to 1), it shall make subscription information known to all standby DMs for that same configuration group provided by that same vendor before responding to the DevMgtSet(InformInfo).

A7.5.7.4 SUBSCRIPTION TIMEOUT

When a subscribed node fails or resets, it does not always have the chance to unsubscribe. For the express purpose of limiting the number of retries and the size of the DM's subscribers list over long periods of time, clients that become unreachable by the DM or otherwise leave the fabric for any reason will have their subscription terminated by the DM. However the DM needs to be able to distinguish between subscribers that are temporarily unreachable verses those that have gone away.

When the DM receives a trap, it sends a DevMgtReport() to each subscriber and waits for a DevMgtReportResp(). If it fails to receive the response, it resends the DevMgtReport(). The minimum amount of time the DM waits before re-sending is specified in the InformInfo attribute. The DM should implement the Retry-Backoff Policy specified in Annex A1 section A1.3.2 between successive attempts to limit the number of retries when trying to reach an unreachable node. If after 10 minutes the subscriber has not responded, the DM may terminate the retry and cancel all of the subscriptions for that client.

CA7-24: If the DM fails to receive a ReportResp() in response to a Report(), then the DM shall continue to retry the Report() until either it receives a ReportResp(), the client unsubscribes, or the DM terminates the subscription.

CA7-25: The DM shall not terminate a subscription due to a missing ReportResp() until after 10 minutes of retrying has occurred. If the DM does terminate a subscription due to a missing ReportResp(), it shall terminate all subscriptions for that particular destination (LID/GID + QPN).

A7.5.7.5 HEARTBEAT

The DM periodically sends a DevMgtReport(Notice=Heartbeat) to let subscribers know that the DM still exists and that the client's subscriptions are still valid. It is also a means for the DM to detect stale subscriptions since failure for a client to respond to the heartbeat is grounds to cancel that client's subscriptions.

When a client subscribes for the heartbeat event, the DM sends the client a DevMgtReport(Notice=Heartbeat)[43] before it replies to the subscription request. This initial heartbeat provides the client with the maximum time until the next heartbeat (TTNH). The length of time between heartbeats is a DM policy. The client starts an expiration timer for TTNH minutes and if the timer expires, it is an indication that the DM is no longer functional or that the DM has cancelled the client's subscriptions. Thus, the client should re-subscribe or query the SA to locate the new DM so it can subscribe. Each time a client receives a DevMgtReport(Notice=Heartbeat), it resets its expiration timer to the new TTNH value specified in the Heartbeat notice.

CA7-26: For heartbeat subscriptions, the DM shall generate a DevMgtReport(Notice=**Heartbeat**) to the subscriber before sending the DevMgtGetResp(InformInfo)

CA7-27: The DM shall continue to generate DevMgtReport(Notice=**Heartbeat**) to each subscriber within TTNH minutes after sending the previous DevMgtReport(Notice=Heartbeat).

Note that the LID Range and GID in the InformInfo for a heartbeat report is irrelevant and thus ignored by the DM. That is, the DM generates a Heartbeat report to anyone subscribed for heartbeats regardless of the GID and LID range.

The Heartbeat notice also has a 'Fail-Over' bit that indicates a new manager. This is to inform the client that a standby manager has gracefully taken over the DM responsibility. In order to ensure that the notice came from a valid DM, instead of using the Report()'s reciprocal path to communicate with the new DM, the client should query the SA to locate the new DM, but it will not need to re-subscribe.

CA7-28: The DM shall not set the FailOver bit in a Heartbeat notice unless the DM inherited all of the subscriptions from the previous manager.

CA7-29: The DM shall only set the FailOver bit in the first Heartbeat notice it sends to a client. Thus, the DM shall not set the FailOver bit in subsequent reports to a client after it receives the ReportResp() from that client.

43. For details of the heartbeat notice see Annex A8: Device Management section A8.3.3.2.5 "Heartbeat Notice" on page 1552.

A7.5.8 GRACEFUL HOT REMOVAL

The IOU perspective of graceful removal of an I/O-Module[44] is specified in Annex A8: Device Management section A8.6 "IOC Graceful Hot Removal" on page 1618. Removal can be initiated either by the IOU (Figure 301 on page 1464), such as an operator pushing a release button, or by the configuration manager (Figure 302 on page 1464), such as an administrator requesting the removal via the configuration manager. Graceful refers to the fact that the DA notifies client platforms affected[45] by the removal prior to indicating “OK to Remove”.

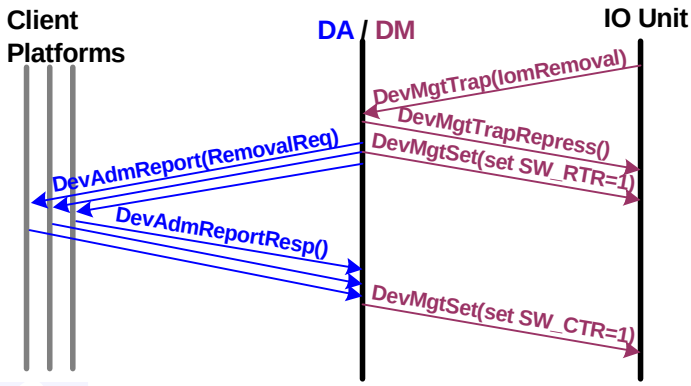


Figure 301 IOU Initiated Removal Process

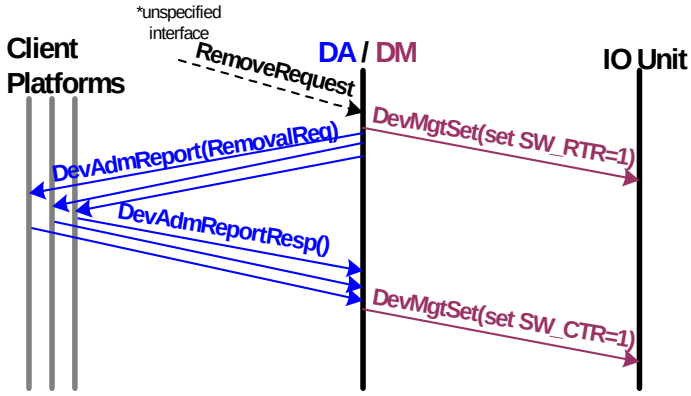


Figure 302 SW Initiated Removal Process

44. Note that this section addresses removal of an I/O module from an IOU where the IOU’s Channel Adapter and Device Management Agent remain functional. The case of removing an IOU including its channel adapter is covered under BaseBorad Management as specified in Volume 2.

45. An affected client is a client platform that has registered with the DA and has access to one or more of the module’s service objects.

For each I/O module, there are 2 bits in the IOU that the DM controls to facilitate graceful hot removal - see Annex A8: Device Management section A8.3.3.9 "SlotControlStatus" on page 1570. They are SW_RTR (software request-to-remove) and SW_CTR (software clear-to-remove). In addition, the module's IOU_RTR bit indicates to the DM that the IOU has initiated the removal request for that I/O module.

Before the DM assigns any of an IOU's service objects to a client (i.e., creates or modifies platform pool table records), the DM clears both the SW_RTR and SW_CTR bits to 0 for all I/O modules, indicating that the modules are in use. This also turns the I/O module's Status LED on indicating that the module is in-use. Until the DM does this, the module's Status LED is off, indicating that it is "OK to Remove". The DM modifies these bits via the DevMgtSet(SlotControlStatus:RemovalControl) as specified in the Device Management Annex.

CA7-30: The DM shall clear the SlotControlStatus.SW_RTR bit and the SW_CTR bit of each I/O module of an IOU before it assigns any of the service objects associated with that I/O module to a client platform. Note than an I/O module might contain multiple IOCs, thus this pertains to all service objects of all IOCs associated[46] with the I/O module.

For the case that the request is coming from the IOU (e.g., operator pressing the I/O module's removal button), the DevMgt agent sets IOU_RTR for the affected module and then forms and sends a DevMgtTrap(IomRemoval) out each port where the port's ClassPortInfo trap information has been set (see A7.6.1.2.9 Removal Requested Notification on page 1477). On setting the IOU_RTR, the DevMgt agent blinks the module's Status LED.

Upon receipt of a DevMgtTrap that indicates IOU_RTR is set, the DM issues a DevMgtSet(SlotControlStatus:RemovalControl=0x02) which instructs the IOU to set the SW_RTR bit to 1b. The DevMgtSet() acts as an acknowledgement to the setting of IOU_RTR. The DA notifies the affected clients via the DevAdmReport(RemovalReq) and collects the necessary responses via the DevAdmReportResp(RemovalReq).

CA7-31: The Configuration Manager shall examine DevMgtTrap()s and shall send a DevMgtSet(SlotControlStatus:RemovalControl=*Set SW_RTR to one*) to the IOU when it receives a DevMgtTrap(IomRemoval).

For the case that the DM initiates the removal, the DM issues a DevMgtSet(SlotControlStatus:RemovalControl=0x02 - '*Set SW_RTR to one*') for the affected I/O module in conjunction with notifying and col-

46. Any IOC with an IOControllerProfile:SlotNumber equal to the I/O module's slot number.

lecting the necessary responses from the affected clients. The RemovalControl value of 0x02 instructs the DevMgt agent to indicate the "transition" condition through the blinking of the I/O module's Status LED.

Regardless of whether the IOU or the configuration manager generated the removal request, the DM sets the ***SlotControlStatus:RemovalControl*.SW_RTR** bit of the I/O-Module to 1 via a DevMgtSet(SlotControlStatus:RemovalControl=*Set SW_RTR to one*). This clears the trap condition (if it was IOU initiated) and starts the LED blinking as a physical indication that the removal approval process is in progress. The DA also issues a DevAdmReport(RemovalReq) to any client platform that has subscribed for removal events and had been assigned a service object that will be affected by the removal.

CA7-32: The Configuration Manager shall initiate the removal process (i.e., sends DevAdmReport(RemovalReq) to subscribed client platforms) when it sends a DevMgtSet(SlotControlStatus:RemovalControl.*Set SW_RTR to one*) to the IOU.

CA7-33: The DA shall issue a DevAdmReport(RemovalReq) to each client platform that has been assigned a service object that will be affected by the removal, if the client platform subscribed for *Removal reports* and if a GID of the IOU matches the InformInfo:GID or the IOU has a LID that falls in the range of the InformInfo: LIDRangeBegin – LIDRangeEnd (see Table 437 DevAdm LID Range Interpretation on page 1491).

An IORM can register for Removal reports via the DevAdmSet(Login) or a DevAdmSet(Informinfo). Refer to A7.6.3.4 LogIn on page 1491 and A7.6.3.3 InformInfo on page 1490.

The DevAdmReport(RemovalReq) sent to a client platform indicates the IOCs that are affected by the removal that have one or more service objects assigned to that client platform. Each DevAdmReport(RemovalReq) can specify up to 16 IOCs and if there are more than 16 affected IOCs for that client platform, then the DA issues multiple announcements to that client platform.

CA7-34: The DevAdmReport(RemovalReq) shall only list IOCs which have service objects listed in a PlatformPoolRecord for which the client platform has been given the Supervisor_Key or listed in a ClientPoolRecord for which the client platform has been given the Client_Key.

Each client platform responds with a DevAdmReportResp(RemovalReq) for each DevAdmReport(RemovalReq) it receives to acknowledge receipt of the report and to accept, reject, or request more time. Based on these responses the Configuration Manager decides whether to approve, delay, or cancel the removal process.

In the DevAdmReport(RemovalReq), the DA indicates the severity of the removal request by setting the RemovePriority field, which indicates if it is a:

- **Graceful Removal** - any IORM may terminate the removal process by rejecting the request, or may request more time so it can stop using the resource before it approves the request. An IORM indicates approval, rejection, or delay via setting the Acknowledge component in the DevAdmReportResp(RemovalReq).
- **Forced Removal** - IORM cannot reject the request, but may request more time in order to stop using the resource before it approves the request.
- **Immediate Removal** - IORM cannot reject nor delay the removal process. The removal continues after the DA has notified all the affected clients.
- **Critical Removal** - removal already approved, thus, the DevAdmReport(RemovalReq) is just a notice sent after the OK-to-Remove was issued.

For Graceful and Forced after all of the affected client platforms have accepted, or for Immediate or Critical, the DM continues the removal process by setting the I/O module's SW_CTR bit to 1. This indicates that the I/O module is no longer in use and turns the Status LED off indicating that it is "OK to Remove".

However, if the Configuration Manager decides to terminate the removal process, the DM issues a DevMgtSet(SlotControlStatus:RemovalControl=0x01) to clear the SW_RTR and IOU_RTR bits and the DA sends an IOC On-Line notice to the set of client platforms to which it sent the RemovalReq report.

If the DA does not receive a DevAdmReportResp() from each client platform within the time identified by that platform's DevAdmSet(InformInfo), the DA repeats the DevAdmReport() to that platform. If a client platform responds with an Acknowledge value = WAIT and the RemovePriority component of the RemovalRequest attribute permits, then the DA resends the DevAdmReport(RemovalRequest) to each client that indicated WAIT until that client has approved. The amount of time the DA should wait before resending is indicated in the DevAdmReportResp.

The exact behavior of the Configuration Manager, as well as its control actions with the I/O-Module being removed is not specified if, for example, the DA does not receive all the anticipated responses back from the client platforms within a reasonable amount of time (Configuration Management policy).

Note that as per the Device Management Annex, if the Configuration Manager fails to respond to the IOU in a timely fashion, hardware time-outs might override and permit the removal anyway.

A7.5.9 DIAGNOSTICS

The configuration management framework provides for a Diagnostic Application to notify and obtain permission from multiple users of an IOU, such that the users participate in determining if a diagnostic can be run at that time. The actual Diagnostics that are run are outside the scope of this Annex and are normally vendor specific.

A7.5.9.1 DIAGNOSTIC FRAMEWORK

As illustrated in Figure 303, the diagnostic framework provides the means for a diagnostic application located on any node to request a diagnostic session. This request is sent to the DM [step 1]. The manager validates the request, and if the source is authorized to perform diagnostics, the DA informs the client platforms that will be affected by the diagnostics [steps 2,3]. If the client platforms agree, the manager establishes a diagnostic session with the IOU [step 4], and then informs the diagnostic application that it may proceed [step 5].

The diagnostic application performs diagnostics directly with the IOU [step 6] and when it is done running diagnostics, it releases the session by informing the DM [step 7]. The DM cancels the IOU's diagnostic session [8] and then the DA notifies the affected clients that the I/O resources are back on-line [9].

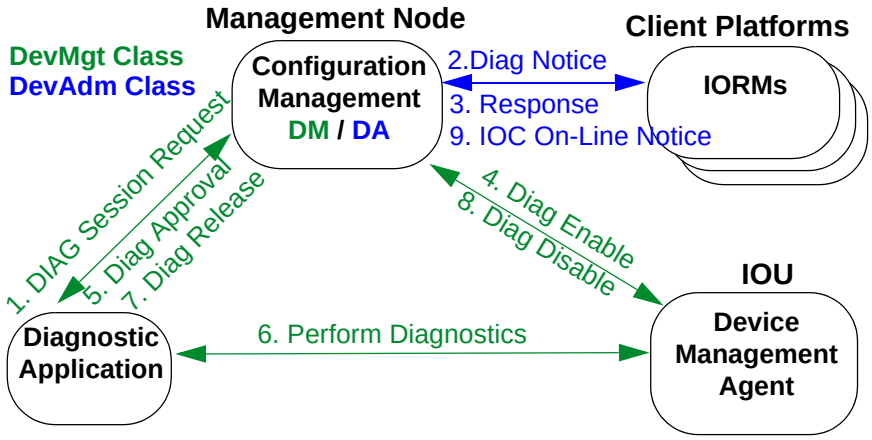


Figure 303 Diagnostic Usage Model

Steps 1,4,5,6,7, and 8 are performed under the DevMgt class and are defined in Annex A8: Device Management. Steps 2,3, and 9 are performed under DevAdm class and are defined in this annex.

oA7-5: If the DM supports Persistent Context (i.e. sets the *IsContextPersistent* bit in DevMgtGetResp(ClassPortInfo) to 1), it shall save Diag Session information in non-volatile storage before sending the DevAdmGetResp(DiagSession).

oA7-6: If the DM supports Graceful Failover (i.e., sets the *GracefulFailover* bit in DevMgtGetResp(ClassPortInfo) to 1), it shall make Diag Session information known to all standby DMs for that same configuration group provided by that same vendor before responding to the DevMgtSet(DiagSession).

CA7-35: Before the DM enables a diagnostic session (i.e., sends the DevMgtSet(DiagSession) to the IOU creating a diag token), the DA shall issue a DevAdmReport(DiagNotice) to each client platform that has been assigned a service object that could be affected by the diagnostic testing (as per the DiagSession:DiagScope), only if the client platform registered for *Diagnostic Notification* and if a GID of the IOU matches the InformInfo:GID or the IOU has a LID that falls in the range of the InformInfo: LIDRangeBegin – LIDRangeEnd.

An IORM can register for *Diagnostic Notification* via the DevAdmSet(Login) or a DevAdmSet(Informinfo). Refer to A7.6.3.4 LogIn on page 1491 and A7.6.3.3 InformInfo on page 1490.

The DevAdmReport(DiagNotice) indicates all of the affected IOCs that are assigned to that client platform. Each DevAdmReport(DiagNotice) can specify up to 16 IOCs and if there are more than 16 IOCs affected, then the DA issues multiple announcements to that client platform.

CA7-36: The DevAdmReport(DiagNotice) shall only list IOCs which have service objects listed in a PlatformPoolRecord for which the client platform has been given the Supervisor_Key or listed in a ClientPoolRecord for which the client platform has been given the Client_Key.

The Diag Notice [step 2] provides the means for the manager to indicate the priority of the diagnostics and the response to the Diag Notice [step 3] allows the client platform to reject or delay the start of the diagnostic session.

A7.5.9.2 VERSION 1 DIAGNOSTICS

The diagnostic framework assumes that IOUs have a Device Management Agent implementing Device Management Class version 2. Many of the features for supporting sharing of IOUs were added in version 2 and are not available in DevMgt class version 1. Therefore, a DM is not able to arbitrate and control diagnostic sessions for IOUs that only implement DevMgt class version 1.

When the DM receives a DIAG Session Request that targets an IOU that only supports version 1, the DM rejects the request with MAD Status = 'IncompatibleVersion' (see Annex A8: Device Management section A8.3.1.2 on page 1535). This serves as an indication to the diagnostic application that the IOU implements a down-level DevMgt Agent. It is possible for the diagnostic application to perform diagnostics directly with the IOU under the bounds of DevMgt class version 1 as per Chapter 16 Section 16.3.4.

CA7-37: A DM shall reject a DIAG Session Request for an IOU that does not support DevMgt class version 2 with a status code of 0x41 - 'IncompatibleVersion'.

Typically, operation under class version 1 assumed that either (a) partitioning would only permit one client platform to access an IOU and that diagnostic testing would be executed from that client platform, or (b) client platforms sharing an IOU would coordinate by an unspecified means. Furthermore, the client platform acted as a manager and configured the IOU to send traps directly to the client platform. Note that there was only one generic trap defined for version 1 (i.e., the Ready-to-Test trap).

The diagnostic application should not attempt to configure the IOU to send traps to the diagnostic application, but rather poll the IOU for the Test-Ready state. Thus, if a client platform does configure a version 1 IOU to send traps to the client platform, then the client platform will be notified of the diagnostic (if the IOU supports traps).

A7.5.9.3 7.5.9.3 DIAGNOSTICS UNDER PASSIVE MANAGEMENT

Typically, operation under Passive Management assumes that either (a) partitioning would only permit one client platform to access an IOU and that diagnostic testing will be executed from that client platform, or (b) client platforms sharing an IOU will coordinate by an unspecified means. Operation under passive management does not provide the degree of protection and notification that active management provides.

The diagnostic framework assumes there is a DM that arbitrates Diag Sessions. However, for passive management, there is no DM to arbitrate, control diagnostic sessions, or distribute traps. Instead, the diagnostic application must establish the Diag Session directly with the IOU by sending the DevMgtSet(DiagSession) to the IOU the same as a DM would. The diagnostic application should not attempt to configure the IOU to send traps to the diagnostic application, but rather poll the IOU for the DiagSessionStatus via the DevMgtGet(DiagSession).

A7.6 DEVADM CLASS DEFINITION

This section describes the DevAdm class methods & attributes, operation of the DA, and relationships between the DA and other IBA management entities. Refer to the Device Management Annex for specification of Device Management class.

A7.6.1 OPERATION

Client platforms query the DA using DevAdm class MADs. The DA provides client platforms with information about IOUs and with the supervisor key that the platform uses to configure the IOU.

Client Platforms may subscribe for event notification from the DA. These events include configuration changes such as IOUs and IOCs coming on-line. Methods to subscribe and un-subscribe for traps are facilitated by the DevAdm class using the DevAdmSet(InformInfo) method in Volume 1 Chapter 13 and further defined below.

The DA responsibilities include:

- Providing the IORM within client platforms with configuration information such as list of IOUs, supervisor keys, and client keys.
- Subscribing with the Subnet Administration for IOU arrival and removal events so it can notify client platforms when I/O services come on-line.
- Monitoring DevMgt traps to detect resource changes so it can notify client platforms when I/O services come on-line.
- Allowing client platforms to subscribe for reports of configuration related events, such as I/O services coming on-line and configuration changes.
- Notifying client platforms and coordinating responses for events such as:
  - IO-Module (one or more IOCs) removal requests
  - Diagnostic session requests

A7.6.1.1 QUERY

The DevAdm query subsystem permits the IORM of a client platform to query the DA for:

- a list of IOUs that have service objects assigned to the client platform.
- the Supervisor_Key for each IOU that the IORM uses to find and configure Client Pool Table records allocated to that client platform.

- a list of Client_Keys for each IOU that the IORM may use to subdivide resources.

The MAD header of a DevAdm MAD contains a RequesterID that identifies the client platform. Before a client platform can submit a query to the DA, it must perform a LOGIN to the DA to obtain its RequesterID. **The LOGIN provides the means for the DA to validate the identity of the client platform and then assign it a RequesterID so that the DA does not have to validate every query**.

The query subsystem allows the requester to submit a single request and the DA responds with a set of attribute records. The response may require more capacity for data than that provided by a single MAD. Thus, DevAdm uses the Reliable Multi-Packet Protocol (RMPP) as specified in Chapter 13.

The attributes obtained by the query are in effect records of attributes from the configuration information, filtered for the requesting platform. Actual implementation of internal data structures and how the DA gets configured are outside the scope of this specification.

CA7-38: When a DA receives a DevAdmGet() for an attribute identified as RMPP in Table 428 DevAdm Attribute / Method Map on page 1486, the DevAdmGetResp() shall only contain records for that requester as identified by the RequesterID. If the RequesterID is not valid (i.e., was not assigned, is no longer in use, or is the default RequesterID), the DA shall return zero records and a status of Invalid RequesterID.

If an IORM has a query rejected for invalid RequesterID, it needs to perform a login to acquire a valid RequesterID.

A7.6.1.2 EVENT NOTIFICATION SUBSYSTEM

Device Administration provides an event-notification subsystem that notifies subscribed client platforms about:

- **Configuration Change** - generates a notice when a change is made to the configuration information that the DA provides to the client platform.
- **Resource Allocation Change** - generates a notice when the DM makes a change to an IOU's pool table record (i.e., adds or removes service objects or changes the resources that the client may consume).
- **IOU On-line Status Change** - monitors subnet to detect when an IOU appears or disappears and generates IOC On-line and IOC Off-line reports.

- **IOC On-line Status Change** - monitors DevMgt traps for IO Controller Change and Slot Status Change notices and generates IOC On-line and IOC Off-line reports if an IOC's on-line status changes.
- **Removal Requested** - reports when an I/O module is to be removed (graceful hot plug).
- **Diagnostic Session Requested** - reports when a diagnostics are to be performed.

The DA generates event reports via the DevAdmReport() when it detects or generates any of these conditions.

Each client platform may subscribe for any of these DevAdm notifications. When the DA detects any of these conditions, it generates a DevAdmReport() to subscribed client platforms. The DevAdmReport() uses different attributes depending on what is being reported. The AttributeID in the DevAdmReport() identifies the type of attribute used in the report.

A7.6.1.2.1 EVENT SUBSCRIPTION

A client platform subscribes with the DA for event notification by issuing a DevAdmSet(InformInfo) to the DA as follows:

The *InformInfo.Subscribe* value indicates the nature of the subscription - A value of 1 to subscribe and 0 to un-subscribe. If successful, the DA will report subscribed events to the subscriber via a DevAdmReport() MAD. The *GID* and *LID Range* in the InformInfo attribute identifies for which IOUs the client wants reports. Note that when subscribing and unsubscribing for the Heartbeat, the InformInfo *GID* and *LID Range* is irrelevant and thus the client should set the GID, LIDRangeBegin, LIDRangeEnd in the DevAdmSet(InformInfo) to zero, 0xFFFF, and zero respectively and the DA shall ignore these components.

The sequence of message exchanges for event subscription mechanism is shown in Figure 304.

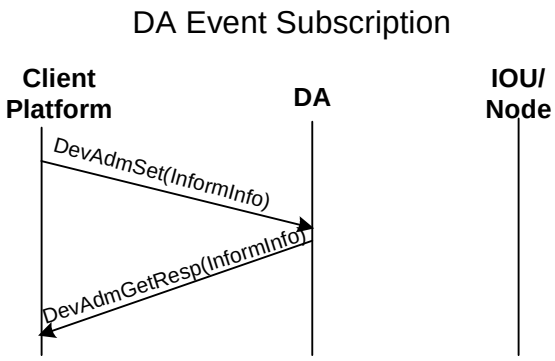


Figure 304 Event Subscription

A Configuration Manager maintains information concerning which service objects may be accessed by which client platforms (assuming that proper partitions exist). An administrator establishes endnode partition memberships through the SM. The Configuration Manager can query the SubnAdm agent (SA) of the SM to validate partition usage (i.e., validate that a path exists between the client and the IOU).

A7.6.1.2.2 EVENT REPORTING

When the DA detects or generates a DevAdm event, it generates a DevAdmReport() to subscribed client platforms. The DevAdmReport() uses different attributes depending on what is being reported. The AttributeID in the DevAdmReport() identifies the type of attribute used in the report. The Removal Requested notification uses the RemovalReq attribute, the Diagnostic notification uses the DiagNotice attribute, the Reset notification uses the ResetNotice attribute, and the others use the Notice attribute. See A7.6.3.2 "Notice" on page 1487 for a list DA generated notice attributes. Diagnostic, Removal Requested, and Reset notification report attributes are as per A7.6.3.7 "RemovalReq" on page 1497, A7.6.3.8 "DiagNotice" on page 1499, and A7.6.3.9 "ResetNotice" on page 1501.

The client responds to the DevAdmReport() with a DevAdmReportResp().

CA7-39: A subscribed client platform shall respond with a DevAdmReportResp() each time it receives a DevAdmReport(). The response shall be within the time frame specified in the DevAdmSet(InformInfo).

If the DA does not receive the DevAdmReportResp() from a client, it repeats the DevAdmReport() to that client.

A7.6.1.2.3 SUBSCRIPTION INTEGRITY

Client platforms depend on receiving Report()s about configuration changes and thus depend on the DA reporting configuration events to

subscribed clients. However, there are a number of situations that can cause a subscription to be destroyed where the client might not be aware that its subscriptions were destroyed. The situations and considerations described in A7.5.7.3 Subscription Integrity also apply to DevAdm subscriptions. Thus, DevAdm also has a Heartbeat notice (see A7.6.1.2.12 "Heartbeat" on page 1478).

oA7-7: If the DA supports Persistent Context (i.e. sets the *IsContextPersistent* bit in DevAdmGetResp(ClassPortInfo) to 1), it shall retain subscriptions across power cycles (i.e., use subscription information stored in non-volatile storage).

oA7-8: If the DA supports Graceful Failover (i.e., sets the *GracefulFailover* bit in DevAdmGetResp(ClassPortInfo) to 1), it shall share LogIn and subscription information with all standby DAs for that configuration group provided by that same vendor. This includes supplying Standby DAs with LogIn and subscription information when they come on-line and updating standby DAs when LogIn and subscription information changes.

A7.6.1.2.4 SUBSCRIPTION TIMEOUT:

When an event occurs, the DA sends a DevAdmReport() to each subscribed client platform and waits for a DevAdmReportResp(). If it fails to receive the response, it resends the DevAdmReport(). The minimum amount of time the DA waits before re-sending is specified in the InformInfo attribute. For resending the report, the DA should implement the Retry-Backoff Policy specified in Annex A1 section A1.3.2 and if after 10 minutes the client platform has not responded, the DA may terminate the retry and cancel all of the subscriptions for that client.

CA7-40: If the DA fails to receive a ReportResp() in response to a Report(), then the DA shall continue to retry the Report() until either it receives a ReportResp(), the client unsubscribes, or the DA terminates the subscription.

CA7-41: The DA shall not terminate a subscription due to a missing ReportResp() until after 10 minutes of retrying has occurred. If the DA does terminate a subscription due to a missing ReportResp(), it shall terminate all subscriptions for that particular destination (LID/GID + QPN).

A7.6.1.2.5 CONFIGURATION CHANGE NOTIFICATION

A configuration change notification results because the DM creates a Pool Table record in an IOU and then assigns that record to the client platform (i.e., provides the client platform with the Supervisor_Key or Client_Key).

When a client platform comes on-line it queries the DA to get a list of IOUs and Supervisor_Keys. When that list changes, the DA informs the client

by issuing a Configuration Change notice to the client platform, if the client subscribes for the Configuration Change Notification.

CA7-42: The DA shall generate a DevAdmReport(Notice=**Configuration Change**) to a subscribed client platform anytime that an S_KeyInfo attribute record is added or removed from the list of records that would be returned in response to a DevAdmGet(S_KeyInfo) from that platform if a GID of the IOU in the S_KeyInfo attribute matches the InformInfo:GID or that IOU has a LID that falls in the range of the InformInfo: LIDRange-Begin – LIDRangeEnd.

CA7-43: The DA shall generate a DevAdmReport(Notice=**Configuration Change**) to a subscribed client platform anytime that a C_KeyInfo attribute record is added or removed from the list of records that would be returned in response to a DevAdmGet(C_KeyInfo) from that platform if a GID of the IOU in the C_KeyInfo attribute matches the InformInfo:GID or that IOU has a LID that falls in the range of the InformInfo: LIDRange-Begin – LIDRangeEnd.

A7.6.1.2.6 IOC ON-LINE NOTIFICATION

The DA generates an IOC on-line notification when it detects an IOU coming on-line or an IOC in an on-line IOU coming on-line.

CA7-44: The DA shall detect when IOUs in its configuration group come on-line by subscribing with the SA for SubnAdm Trap 64.

The DM configures IOUs to send DevMgt traps to the Configuration Manager. When the Configuration Manager receives a DevMgt Trap, it determines the exact nature of the change, updates its information base as needed, and generates the necessary event notifications.

CA7-45: The DA shall inspect DevMgt Traps for IO Controller Change notices and Slot Status Change notices to detect IOCs coming on-line and going off-line.

CA7-46: The DA shall generate a DevAdmReport(Notice=**IOC On-line**) to subscribed client platforms when it detects the presence (coming on-line) of each IOC that contains service objects assigned to the client platform (i.e., service object is listed in the Platform Pool Table or Client Pool Table record for which the client has the Supervisor_Key or Client_Key) and if a GID of the IOU matches the InformInfo:GID or the IOU has a LID that falls in the range of the InformInfo: LIDRangeBegin – LIDRangeEnd.

A7.6.1.2.7 IOC OFF-LINE NOTIFICATION

CA7-47: The DA shall detect when IOUs in its configuration group go off-line by subscribing with the SA for SubnAdm Trap 65.

As per CA7-45: on page 1476 the DA inspects DevMgt Traps to detect IOCs coming on-line and going off-line.

CA7-48: The DA shall generate a DevAdmReport(Notice=**IOC Off-line**) to subscribed client platforms anytime that it detects that an IOC that contains service objects assigned to the client platform (i.e., service object is listed in the Platform Pool Table or Client Pool Table record for which the client has the Supervisor_Key or Client_Key) went off-line when a GID of the IOU matches the InformInfo:GID or the IOU has a LID that falls in the range of the InformInfo: LIDRangeBegin – LIDRangeEnd.

A7.6.1.2.8 RESOURCE ALLOCATION CHANGE NOTIFICATION

The DA generates a Resource Allocation Change notification when the DM modifies an IOU's pool table record changing which service objects a client platform may use or changing the IOU resources that the client platform is allowed to consume. The DA does not send the report until the DM completes the IOU configuration update - i.e., receives a successful response to the DevMgtSet() as illustrated in Figure 305.

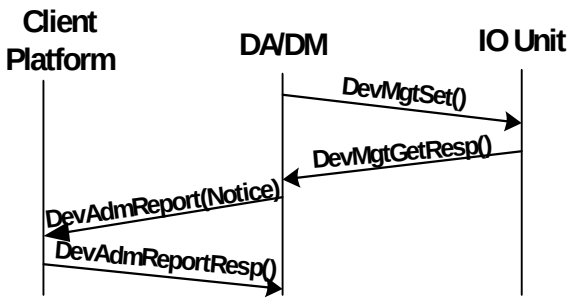


Figure 305 Event Notification for Resource Allocation Change

CA7-49: The DA shall generate a DevAdmReport(Notice=**Resource Allocation Change**) to each subscribed client platform anytime that the DM modifies one or more platform pool table records or client pool table records for that client platform (i.e., the client platform has been given the Supervisor_Key or Client_Key for the pool table record) and if a GID of the IOU matches the InformInfo:GID or the IOU has a LID that falls in the range of the InformInfo: LIDRangeBegin – LIDRangeEnd.

A7.6.1.2.9 REMOVAL REQUESTED NOTIFICATION

The DA generates a Removal Requested notification when it initiates an I/O module removal or receives a removal request from the IOU. The removal requested notification DevAdmReport() does not use the Notice attribute. Rather the DevAdmReport() for this event uses the RemovalReq

attribute (as indicated in the MAD's AttributeID field). See A7.5.8 "Graceful Hot Removal" on page 1464 for details.

CA7-50: The DA shall inspect DevMgt Traps to detect I/O module removal requests (i.e., a hot plug removal request).

The DA generates a DevAdmReport(RemovalReq) to client platforms subscribed for the Removal Request event as per A7.5.8 "Graceful Hot Removal" on page 1464.

A7.6.1.2.10 DIAGNOSTIC NOTIFICATION

The DA generates the Diagnostic Notification event when a diagnostic application requests a diagnostic session that would impact the client platform's I/O operation. The Diagnostic Notification does not use the Notice attribute. Rather the DevAdmReport() for this event uses the DiagNotice attribute, which is indicated in the DevAdmReport's AttributeID field. See A7.5.9 "Diagnostics" on page 1468 for details.

The DA generates a DevAdmReport(DiagNotice) to client platforms subscribed for the Diagnostic event as per A7.5.9.1 "Diagnostic Framework" on page 1468.

A7.6.1.2.11 RESET NOTIFICATION

The DA generates the Reset Notification event before it issues a DevMgtSet(Reset) to an IOU that would impact the client platform's I/O operation. The Reset Notification does not use the Notice attribute. Rather the DevAdmReport() for this event uses the ResetNotice attribute, which is indicated in the DevAdmReport's AttributeID field. See A7.6.3.9 "ResetNotice" on page 1501 for details.

A7.6.1.2.12 HEARTBEAT

The DA periodically sends a DevAdmReport(Notice=Heartbeat) to let subscribed client platforms know that the DA still exists and that the platform's subscriptions are still valid. It is also a means for the DA to detect stale subscriptions since failure for a platform to respond to the heartbeat is grounds to cancel all subscriptions for that platform.

When a client subscribes for the heartbeat event, the DA immediately sends it a DevAdmReport(Notice=Heartbeat). This initial heartbeat provides the client with the time until the next heartbeat (TTNH). The length of time between heartbeats is a DA policy. The client starts an expiration timer for TTNH minutes and if the timer expires, it is an indication that the DA is no longer functional or that the DA has cancelled the platform's subscriptions. Thus, the client platform should re-subscribe (in the case of subscription cancellation). If the DA does not respond, then the client should query the SA to locate the new DM (in case the manager reset or

a standby manager has taken over) and query the new DM to locate the DA so it can login and subscribe.

Each time a client receives a DevAdmReport(Notice=Heartbeat), it resets its expiration timer to the new TTNH timeout value specified in the Heartbeat notice. The Heartbeat notice also has a bit that indicates a new manager. This is to inform the client that a standby manager has gracefully taken over the DM / DA responsibility. The client will need to query the SA to locate the new DM and query the new DM to locate the DA, but it will not need to login nor re-subscribe.

IORMs can subscribe for the Heartbeat notice either explicitly via DevAdmSet(Informinfo) or implicitly as a result of DevAdmSet(Login).

CA7-51: For explicit heartbeat subscriptions, the DA shall generate a DevAdmReport(Notice=**Heartbeat**) to the subscribed client platform before sending the DevAdmGetResp(InformInfo)

CA7-52: .For implicit heartbeat subscriptions, the DA shall generate a DevAdmReport(Notice=**Heartbeat**) to the subscribed client platform before sending the DevAdmGetResp(LogIn).

CA7-53: The DA shall continue to generate DevAdmReport(Notice=**Heartbeat**) to each subscribed client platform within TTNH minutes after sending the previous DevAdmReport(Notice=Heartbeat).

CA7-54: The DA shall not set the FailOver bit in a Heartbeat notice unless the DA inherited all of the subscriptions from the previous manager.

CA7-55: The DM shall only set the FailOver bit in the first Heartbeat notice it sends to a client. Thus, the DM shall not set the FailOver bit in subsequent reports to a client after it receives the ReportResp() from that client.

Note that the LID Range and GID in the subscription [i.e., from the DevAdmSet(InformInfo)] for a heartbeat notice is irrelevant and thus ignored by the DA. That is, the DA sends the heartbeat to all clients who subscribed for the Heartbeat regardless of the GID and LID Range in the InformInfo.

A7.6.2 DEVADM MESSAGE FORMAT

Figure 306 shows the structure of a DevAdm datagram. The DA sends and receives datagrams using an Unreliable Datagram QP, that is determined via the CM:SIDR_REQ protocol. The datagrams conform to the MAD definition as specified in Volume 1 Chapter 13

.

Figure 306 DevAdm MAD Format

Offset	Byte 0	Byte 1	Byte 2	Byte 3
0	Common MAD Header			
...				
20				
24	RMPP Header			
...				
32				
36	RequesterID			
40				
44	reserved			
...				
56				
60	reserved		ComponentMask	
64	DevAdm Data			
...				
252				

CA7-56: A DA and IORM shall send and receive DevAdm class MADs that conform to the format specified in Figure 306 DevAdm MAD Format and Table 424 Device Administration MAD Fields.

Table 424 defines the fields in the DevAdm datagram.

Table 424 Device Administration MAD Fields

Field[a]	Length	Description
Common MAD Header	24 bytes	Common MAD Header as described in 13.4.2 Management Datagram Format on page 718. MgtClass=0x10, ClassVersion=1.
RMPP Header	12 bytes	RMPP header as described in 13.4.2 Management Datagram Format on page 718. Also refer to A7.6.2.4 RMPP Header on page 1482
RequesterID	8 bytes	A value assigned by the DA to identify the requester. Default value is zero. See A7.6.2.5 RequesterID on page 1483 and A7.6.3.4 LogIn on page 1491.
reserved	16 bytes	reserved
reserved	2 bytes	reserved

Table 424 Device Administration MAD Fields (Continued)

Field[a]	Length	Description
ComponentMask	2 bytes	Indicates which attribute components in the DevAdmGet() are to be used for the query. Refer to A7.6.2.6 Component Mask on page 1484
DevMgt Data	192 bytes	192 bytes of DevAdm payload. The structure and content depends upon the Method, Attribute, and Attribute Modifier fields in the MAD header.

a. The term 'MAD header' refers to all fields except the DevAdm Data field

A7.6.2.1 RESERVED FIELDS

Fields indicated as "reserved" shall be set to zero in request messages. In response messages, a reserved field may be left unmodified if using the request for the response, or may be set to zero. A reserved field shall be ignored on receipt.

CA7-57: Fields indicated as "reserved" shall be set to zero in request messages. In response messages, a reserved field shall either be left unmodified, if using the request for the response, or shall be set to zero.

CA7-58: The recipient of a DevAdm class MAD shall ignore reserved fields.

A7.6.2.2 DEVADM STATUS VALUES

Chapter 13 defines bits 0-7 of the MAD Header.Status field and leaves bits 8-15 for additional class specific status. They are encoded as follows:

Table 425 DevAdm MAD Status Field Values

Name	Bits	Meaning
DA_ERR_CODE	8-11	An enumerated value as follows: 0x0 = normal 0x1 = RequesterID is invalid 0x2 = Supplied request or update is invalid 0x3 = Insufficient privilege or policy violation 0x4 = Not Master DA all other values reserved
RESERVED	12-15	reserved for future definition

CA7-59: If the DA accepts a DevAdmGet() as a valid request, it shall set the MAD Header Status to zero in the DevAdmGetResp().

CA7-60: If the DA rejects a DevAdmGet(), it shall set the MAD Header Status in the DevAdmGetResp() to reflect the reason for the rejection.

A7.6.2.3 METHODS

CA7-61: A DA shall support the DevAdm methods described in Table 426.

Table 426 DevAdm Methods

Method Type	Value	Description
DevAdmGet( )	0x01	Request (read) an attribute.
DevAdmSet( )	0x02	Request a set (write) of an attribute.
DevAdmGetResp( )	0x81	The response from a DevAdmGet or DevAdmSet.
DevAdmReport( )	0x06	Forward a subscribed event.
DevAdmReportResp( )	0x86	Reply to a DevAdmReport.

DevAdmGet( ), DevAdmSet( ), DevAdmGetResp( ), DevAdmReport( ), & DevAdmReportResp( ) methods are common methods specified in Volume 1 Chapter 13 (see Get, Set, GetResp, Report, & ReportResp in section 13.4.5).

The DevAdmSet(InformInfo), DevAdmGetResp(InformInfo), DevAdmReport( ), & DevAdmReportResp( ) methods are used to subscribe to and report DevAdm events. The DevAdmReport( ) & DevAdmReportResp( ) typically use the Notice attribute (A7.6.3.2 “Notice” on page 1487). However, for reporting the IO-Module removal process, the RemovalReq attribute is used in the DevAdmReport( ) & DevAdmReportResp( ) as per A7.5.8 “Graceful Hot Removal” on page 1464. For reporting Diagnostic, the DiagNotice attribute is used in the DevAdmReport() & DevAdmReportResp(). For reporting IOU/IOC Reset, the ResetNotice attribute is used in the DevAdmReport() & DevAdmReportResp().

A7.6.2.4 RMPP HEADER

DevAdm uses the Reliable Multi-Packet Protocol as described in Section 13.6 “Reliable Multi-Packet Transaction Protocol”. This protocol allows the initiator to make a single query which returns more information than can be transferred in a single packet. For example, an IORM can make a single request for S_KeyInfo attributes and have the DA return all of the records for that particular client platform. The response includes multiple DevAdmGetResp(S_KeyInfo) packets if necessary. Only the method/attribute combinations listed as ‘RMPP’ in Table 428 "DevAdm Attribute / Method Map" on page 1486 use the Reliable Multi-Packet Protocol and thus have multi-packet responses. DevAdmGet() requests are always single packets.

CA7-62: A DA shall implement Reliable Multi-Packet Protocol as described in Section 13.6 “Reliable Multi-Packet Transaction Protocol.

CA7-63: A DA shall not respond with more than one packet for method/attribute combinations not listed as 'RMPP' in Table 428 "DevAdm Attribute / Method Map" on page 1486.

CA7-64: A DA shall reject a request that contains more than one packet.

A7.6.2.5 REQUESTERID

Since it is possible that a client platform might use multiple ports and because a port's LID/GID can subsequently change, depending on LID/GID alone for identifying the platform is not sufficient, thus the DA provides each client platform with a RequesterID, which the client platform uses in future requests.

RequesterID serves as an authorization tool. When the DA encounters a new client platform, it validates the client platform and assigns it a RequesterID as follows:

- When a client platform initializes, it uses the default MADHeader:RequesterID value of zero and performs a LOGIN (i.e., sends a DevAdmSet(LogIn) to the DA. In the response, the DA provides a unique RequesterID that the client platform uses in the MADHeader:RequesterID of subsequent DevAdm queries (see A7.6.3.4 "LogIn" on page 1491).
- When the DA receives a DevAdmSet(LogIn), it validates the client platform. At a minimum the DA relates the LID/GID in the request to a specific platform and determines the access rights for that platform. Note that the DA can issue a SubnAdmGet(NodeInfo) to the SA to get the NodeGUID for that platform.

CA7-65: The DA shall identify a client platform by its NodeGUID.

CA7-66: The DA shall validate a client's NodeGUID by querying the SA before sending the DevAdmGetResp(LogIn).

CA7-67: Generation of RequesterIDs shall not be sequential or predictable, in order to make it difficult for a client to guess the RequesterID of another node.

The DA uses the RequesterID to validate DevAdm class requests that it receives.

CA7-68: A DA shall reject a DevAdmGet() containing an invalid RequesterID (see CA7-38: on page 1472).

The use of the default RequesterID by a client platform is only permitted for reading the ClassPortInfo, performing a login, and responding to a DevAdmReport(). The DA uses the default RequesterID in DevAdmReport()s to allow reports to be sent to clients that don't need to know the

platforms's RequesterID. The fact that the IORM used the RequesterID to perform the subscription and the DevAdmReportResp() contains the TransactionID is sufficient protection.

CA7-69: A DA shall only allow the default MADHeader:RequesterID value of zero in DevAdmGet(ClassPortInfo), DevAdmSet(LogIn), and DevAdmReportResp() MADs.

CA7-70: A DA shall use the default MADHeader:RequesterID value of zero in DevAdmReport() MADs that it generates.

It's possible that the information a DA returns in response to a query depends on the state of the client platform (e.g., pre-boot or post boot) and the platform's OS. For example, a booting platform might only be given information about devices necessary to boot the OS, but when the OS takes control, it is given information about additional resources. Since this information can be determined as part of the login process, the RequesterID can function as a data versioning key. When there is a context change, the client platform performs another login. If the DA detects changes (such as a change in the platform's purpose or permissions), it can invalidate the platform's old RequesterID, forcing the client platform to perform another login and get a new RequesterID.

A7.6.2.6 COMPONENT MASK

The ComponentMask in the MAD header is used in queries for attributes supporting RMPP (see Table 428 DevAdm Attribute / Method Map on page 1486). The ComponentMask allows the initiator to indicate which components in the query (i.e., the DevAdmGet request) that the DA uses in determining which records to return. Each bit corresponds to an attribute component as specified in each attribute's definition in A7.6.3: Attributes. If that bit is set to zero, the DA ignores that component when selecting which attributes to return. When the bit is one, the DA only returns records that have that component matching the value supplied in the request. For example, a DevAdmGet(C_KeyInfo) with component mask bits for IouGUID and Supv_Key set to one will return all records matching the IouGUID and Supv_Key values supplied in the attribute in the DevAdmGet(C_KeyInfo) query.

CA7-71: The DA shall ignore MADHeader:ComponentMask in MADs not indicated as 'RMPP' in Table 428 "DevAdm Attribute / Method Map" on page 1486

CA7-72: The DA shall ignore MADHeader:ComponentMask bits that are not defined in the attribute's definition in A7.6.3: Attributes

CA7-73: For method/attribute combinations marked 'RMPP' in Table 428 DevAdm Attribute / Method Map on page 1486, the DA shall only return

attribute records matching the components indicated by the request's MADHeader:ComponentMask field.

CA7-74: The DA shall set the MADHeader:ComponentMask in the response to the same value as in the request, except that any ComponentMask bits not supported (i.e., not defined for the attribute) shall be set to zero. This requires that the MADHeader:ComponentMask in responses to non RMPP method/attributes combinations is always returned as zero.

A7.6.2.7 LOST MESSAGES

The DevAdm *ClassPortInfo* attribute provides a time-out value (RespTimeValue) to be used in conjunction with DevAdm requests. Because DevAdm messages use unreliable datagram service, it is possible that a request or its response may be lost. If the requester does not receive a response within the time allotted[47], it resends the original request.

Where a response is a multi-packet segmented response, the segments are sent in order and identified by the SegmentNumber field. If the initiator detects that it missed one of the response packets, it can immediately request retransmission starting with the missing packet by resubmitting the original request with original RequesterID, setting the ReSend Request bit in the FragmentFlag field, and specifying the missing packet in the SegmentNumber field. The DA starts resending the response starting with the indicated packet using the window value specified in the Resend Request.

A7.6.3 ATTRIBUTES

Table 427 lists attributes for the DevAdm class MADs

Table 427 DevAdm Attributes

Attribute Name	Attribute ID	Attribute Modifier	Description
ClassPortInfo	0x0001	0x00000000	Class Info as per base MAD definition. This document specifies DevAdm ClassVersion =1
Notice	0x0002	0x00000000	Fabricated Trap/Notice information generated by the DA.
InformInfo	0x0003	0x00000000	Subscription information as per common MAD attribute definitions in Chapter 13.
LogIn	0x0011	0x00000000	Provides information about the Client Platform so the DA can assign a RequesterID.
S_KeyInfo	0x0012	0x00000000	Request for list of IOU's and associated Supervisor_Keys.
C_KeyInfo	0x0013	0x00000000	Request for list of Client_Keys.

47. Note that the requester must also consider fabric round trip delays.

Table 427 DevAdm Attributes (Continued)

Attribute Name	Attribute ID	Attribute Modifier	Description
RemovalReq	0x0021	0x00000000	Notifies client platform about IOCs affected by a requested removal of an I/O module.
DiagNotice	0x0022	0x00000000	Notifies client platforms affected by a requested diagnostic.

Table 428 associates DevAdm class attributes with methods.

Table 428 DevAdm Attribute / Method Map

Attribute	Get / GetResp[a]	Set / GetResp	Report / ReptResp
ClassPortInfo	SP		
Notice			SP
InformInfo		SP	
LogIn		SP	
S_KeyInfo	RMPP		
C_KeyInfo	RMPP		
RemovalReq			SP
DiagNotice			SP

a. SP designates single packet request/response. RMPP designates that the method/attribute uses Reliable Multiple Packet Protocol. Note that the DevAdmGet() is always a single packet that can result in a multiple packet response.

A7.6.3.1 CLASSPORTINFO

The ClassPortInfo attribute is defined in Volume 1 Chapter 13 and has the following class specific information and definitions:

- ClassVersion: This is version 1 of the DevAdm class specification. The ClassVersion field shall be set to 0x01.
- CapabilityMask component is defined in Table 429 on page 1487.

.

Table 429 Device Administration ClassPortInfo:CapabilityMask

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.8.1 ClassPortInfo on page 734. Note that bits 0 and 1 are not meaningful for a manager and thus are set to zero and ignored.
8-13	reserved	reserved
14	GracefulFailover	This bit indicates if the DA shares subscription information with standby mangers. A value of 1 indicates that subscriptions are retained across fail-overs. Requirements for setting this bit are specified in section A7.6.1.2.3 Subscription Integrity on page 1474 and A7.6.3.4 LogIn.
15	IsContextPersistent	This bit indicates if the DA persistently stores Login context and subscription information. A value of 1 indicates that subscriptions and login context are retained across reset, restarts, and power cycles. Requirements for setting this bit are specified in A7.6.1.2.3 Subscription Integrity, A7.6.3.3 InformInfo, and A7.6.3.4 LogIn.

A7.6.3.2 NOTICE

The DA uses the Notice attribute in most reports it sends to subscribed client platforms (see below for exceptions). The Notice attribute is defined in Volume 1 Chapter 13 section 13.4.8.2. Note that the DevAdmSet(InformInfo) permits a client platform to subscribe to notices about any IOU in the configuration group. In addition to notice attributes, DevAdm also defines additional attributes that are used in DevAdmReport() MADs (see A7.6.3.7 "RemovalReq" on page 1497 and A7.6.3.8 "DiagNotice" on page 1499).

DevAdm notices are fabricated and fall into two categories - Proxy Traps and Manager Events. A proxy trap is one that is generated on behalf of an IOU, in which case the notice identifies the IOU. A Manager Event is not associated with an IOU.

Unless otherwise specified, components for generic Notice attributes (listed in Table 431 on page 1488) shall be set as per Table 430 on page 1488:

Table 430 Notice Component Values

Component	Component Value
IsGeneric	1 (Generic)
Type	as per Table 431
ProducerType	4 (Class Manager)
TrapNumber	as per Table 431
IssuerLID	• For Proxy Traps - A LID of the subject IOU. • For Manager Events - the base LID of the port emitting the Report() message.
NoticeToggle	0 as per 13.4.8.2 Notice on page 737
Notice Count	0 as per 13.4.8.2 Notice on page 737
DataDetails	as per Table 431
IssuerGID	GID of port specified by IssuerLID

Table 431 Generic DevAdm Events

Trap Number	Name	P/M[a]	Type	DataDetails
0	Removal Requested	P	1-urgent	n/a - the Removal Requested notification does not use the Notice Attribute. It is included here because this event needs a trap number and type so client platforms can subscribe for the notification.
1	Diagnostic Notification	P	1-urgent	n/a - the Diagnostic Notification does not use the Notice Attribute. It is included here because this event needs a trap number and type so client platforms can subscribe for the notification.
2	Configuration Change	P	4-informational	As specified in Table 432
3	IOC On-line	P	4-informational	As specified in Table 433
4	IOC Off-line	P	4-informational	As specified in Table 434
5	Resource Allocation Change	P	4-informational	As specified in Table 435
7	Heartbeat	M	4-informational	As specified in Table 436
8	Reset Notification	P	1-urgent	n/a - the Reset Notification does not use the Notice Attribute. It is included here because this event needs a trap number and type so client platforms can subscribe for the notification.

a. P/M indicates if (P) Proxy Trap or (M) Manager Event.

The DataDetails field of a Notice attribute consists of 54 Bytes that are class and trap specific.

A7.6.3.2.1 CONFIGURATION CHANGE

The DataDetails field for the Configuration Change Notice is defined in Table 432.

Table 432 Configuration Change Notice DataDetails

Component	Length[a]	Description
reserved	64 bits	reserved
IouGUID	64 bits	I/O Unit's Channel Adapter GUID

a. remaining space reserved - set to zero and ignored

A7.6.3.2.2 IOC ON-LINE

The DataDetails field for the IOC On-line Notice is defined in Table 433.

Table 433 IOC On-line Notice DataDetails

Component	Length[a]	Description
reserved	64 bits	reserved
IouGUID	64 bits	I/O Unit's Channel Adapter GUID
IocGUID	64 bits	I/O Controller GUID
SlotNumber	8 bits	Slot number of the I/O module containing the IOC

a. remaining space reserved - set to zero and ignored

A7.6.3.2.3 IOC OFF-LINE

The DataDetails field for the IOC Off-line Notice is defined in Table 434.

Table 434 IOC Off-line Notice DataDetails

Component	Length[a]	Description
reserved	64 bits	reserved
IouGUID	64 bits	I/O Unit's Channel Adapter GUID
IocGUID	64 bits	I/O Controller GUID
SlotNumber	8 bits	Slot number of the I/O module containing the IOC - zero if not known

a. remaining space reserved - set to zero and ignored

A7.6.3.2.4 RESOURCE ALLOCATION CHANGE

The DataDetails field for the Resource Allocation Change Notice is defined in Table 435.

Table 435 Resource Allocation Change Notice DataDetails

Component	Length[a]	Description
reserved	64 bits	reserved
IouGUID	64 bits	I/O Unit's Channel Adapter GUID

a. remaining space reserved - set to zero and ignored

A7.6.3.2.5 HEARTBEAT

The DataDetails field of a Heartbeat Notice attribute is defined in Table 436.

Table 436 Heartbeat Notice DataDetails

Component	Length[a]	Description
TTNH	12 bits	Time till next heartbeat - specifies the number of minutes before the DM will send another Heartbeat notice. If more that this time elapses, it is an indication that the DA has terminated the subscription.
reserved	4 bits	reserved
Fail-over	1	When this bit is set to one, it indicates that a standby manager has taken-over. The client platform should query the SA to locate the new DM.

a. remaining space reserved - set to zero and ignored

A7.6.3.3 INFORMINFO

A client platform uses the InformInfo attribute to subscribe for configuration event notification.

The InformInfo attribute is defined in Volume 1 Chapter 13. Normally, LidRangeBegin and LIDRangeEnd identify the port generating a trap. However, the DA does not forward traps but rather creates configuration events that relate to specific IOUs. Thus, for DevAdm class subscription, the LID range specifies the IOUs involved.

Since the LID range specifies the source LID, which must be in the unicast LID range (0x0001 - 0xBFFF), LidRangeBegin and LIDRangeEnd have class specific interpretation as specified in Table 437.

Table 437 DevAdm LID Range Interpretation

LIDRangeBegin	LIDRangeEnd	Description
0x0000	any	invalid
0x0001 through 0xBFFF	0x0000 and LIDRangeBegin through 0xBFFF	events relating to an IOU managed by the DA/DM that has a LID in the specified LID Range for which the client is assigned a service object.
0xC000 thru FFFE	any	invalid
any	0xC000 thru 0xFFFF	invalid
0xFFFF	any	events relating to any IOU managed by the DA/DM for which the client is assigned a service object and for events not related to an IOU (e.g. Heartbeat).

oA7-9: If the DA supports Persistent Context (i.e. sets the *IsContextPersistent* bit in DevAdmGetResp(ClassPortInfo) to 1), it shall save subscription information in non-volatile storage before responding to a DevAdmSet(InformInfo).

oA7-10: If the DA supports Graceful Failover (i.e., sets the *GracefulFailover* bit in DevAdmGetResp(ClassPortInfo) to 1), it shall make subscription information known to all standby DAs for that same configuration group provided by that same vendor before responding to the DevAdmSet(InformInfo).

A7.6.3.4 LOGIN

A client platform sends DevAdmSet(LogIn) to get its RequesterID and/or to invalidate previous context between the platform and the DA (see A7.6.2.5 "RequesterID" on page 1483). The DA uses the LID/GID in the MAD and R/W information in the LogIn attribute to validate who the requester is and responds back with the LogIn attribute specifying the RequesterID that the platform uses in subsequent queries. As part of a Login, the client platform can request that the DA invalidate all previous event subscriptions made by that platform.

Table 438 LogIn Attribute

Component	Access	Offset (bits)	Length	Description
RequesterID	R/W	0	8 bytes	The value that the client platform uses in MAD-Header:RequesterID of DevAdm queries.
PlatformName	R/W	64	64 Bytes	Indicates the name assigned to the client platform. UTF-8 encoded, null-terminated character string (e.g., “WebServer1”). Value of all zeros indicates unknown or not assigned.
OSName	R/W	576	32 Bytes	Indicates the client platform’s operating system. UTF-8 encoded, null-terminated character string. Value of all zeros indicates unknown or not assigned.
BootStage	R/W	832	2 bits	Indicates the current boot stage of the platform 00 - Pre-boot 01 - OS booting (OS and Pre-boot environments both exist) 10 - OS fully operational (No pre-boot environment)
reserved	RO	834	2 bits	reserved
ContextChange	R/W	836	4 bits	Controls if the DA invalidates existing RequesterIDs and subscriptions made by the client platform as per Table 439 "Context Change" on page 1493
ImplicitSubscription	R/W	840	8 bits	Client sets appropriate bits to automatically subscribe to the indicated event. bit 0 - Removal Request events bit 1 - Diagnostic events bit 2 - Configuration Change events bit 3 - IOC On-line events bit 4 - IOC Off-line events bit 5 - Resource Allocation Change events bit 6 - bit 7 - Heartbeat
reserved	RO	848	688	reserved

See A7.6.2.5 "RequesterID" on page 1483 for requirements on issuing RequesterID. The DA ignores the MADHeader:RequesterID when it receives a DevAdmSet(LogIn) and uses the LID/GID from the MAD to validate who the requester is by getting its NodeGUID from the SA. The Login attribute’s PlatformName, OSName, and BootStage components provides the DA a means to distinguish between different uses of that platform and also as an aid in identifying the platform to an administrator when the platform is not known to the DA. How the DA uses these components is subject to DA policy.

A platform normally issues a new login after the platform is reset, when information in the LogIn attribute (PlatformName, OSName, or BootStage component) changes[48], after it receives a response with a status of invalid RequesterID, or after the operating environment changes (such as the transition from BIOS to OS control) to establish a new context with the DA.

When a client platform initializes, the BIOS, OS, etc., does not necessarily know if context that the previous instantiation of the platform established with the DA is still valid, so the client-platform is able to set the ContextChange component controlling how the DA flushes old context for that platform.

Table 439 Context Change

Bit 3	Bit 2	Bit 1	Bit 0	Action
x	x	x	0	DA does not invalidate any RequesterID.
x	x	x	1	DA invalidates the RequesterID specified in this attribute (i.e., Login:RequesterID) - if Login:RequesterID=zero, then the DA invalidates all RequesterIDs for this platform.
x	x	0	x	DA responds with zero in Login:RequesterID component. DA does not generate any new requester IDs.
x	x	1	x	DA responds with current RequesterID in Login:RequesterID - DA will generate a new RequesterID if one does not exist or it had been invalided.
0	0	x	x	DA does not invalidate or modify any subscriptions.
0	1	x	x	DA invalidates all subscriptions for the RequesterID specified in Login:RequesterID - if Login:RequesterID=zero, then the DA invalidates all subscriptions for all RequesterIDs for this platform.
1	0	x	x	IORM inherits subscriptions for the RequesterID specified in Login:RequesterID, if Login:RequesterID=zero, then it inherits all subscriptions of all RequesterIDs for this platform. This means the DA modifies those subscriptions such that the Report()'s LRH:DLID, LRH:SL, BTH:P_Key, BTH:DestinationQP, DETH:Q_Key, GRH:DGID, GRH:FlowLabel, GRH:TClass, and GRH:HopLmt will be the same values as the values in the DevAdmGetResp(Login).
1	1	x	x	invalid, DA rejects a Set() with this value.

A node that knows it is leaving the fabric (e.g., being put to sleep) can destroy its subscriptions by sending a DevAdmSet(LogIn) with context = 0x4 (or 0x5 to invalidate its RequesterID). When the node awakes, it sends a

48. An example is a platform that changes from a web server to a backup server at midnight and changes back to a web server at 6 AM.

DevAdmSet(LogIn) with context = 0x2 to get its RequesterID and, if desired, make implicit subscriptions.

CA7-75: The DA shall reject a DevAdmSet(LogIn) that specifies a Login:RequesterID component not valid for the client platform as determined by the platform's NodeGUID (see CA7-65: and CA7-66: on page 1483).

CA7-76: Upon receipt of a DevAdmSet(LOGIN) with a ContextChange value that invalidates RequesterID, the DA shall reject subsequent DevAdm MADs that use the invalidated ReqesterID(s).

A client platform uses the Login:RequesterID returned in the DevAdmGetResp(LogIn) in the MADHeader:RequesterID in all subsequent DevAdm MADs.

CA7-77: A client platform shall only use the default RequesterID in DevAdmGet(ClassPortInfo), DevAdmSet(LogIn), and DevAdmReportResp() MADs.

The ImplicitSubscription component provides a shortcut for the client platform to subscribe with the DA for reports. When the client sets the implicit subscription, the DA treats it as receiving an InformInfo for the specified notices. The DevAdmReport()s will be sent to the same destination as the DevAdmGetResp(Login). That is, the values for LRH:DLID, LRH:SL, BTH:P_Key, BTH:DestinationQP, DETH:Q_Key, GRH:DGID, GRH:FlowLabel, GRH:TClass, and GRH:HopLmt are inherited from the values in the DevAdmGetResp(Login). The implicit scope is "all IOUs" (i.e., LIDRangeBegin=0xFFFF).

oA7-11: If the DA supports Persistent Context (i.e. sets the *IsContextPersistent* bit in DevAdmGetResp(ClassPortInfo) to 1), it shall save/update Login information in non-volatile storage before responding to a DevAdmSet(LogIn).

oA7-12: If the DA supports Graceful Failover (i.e., sets the *GracefulFailover* bit in DevAdmGetResp(ClassPortInfo) to 1), it shall make LogIn information known to all standby DAs for that same configuration group provided by that same vendor before responding to a DevAdmSet(LogIn).

A7.6.3.5 S_KEYINFO

The DA responds to *DevAdmGet(S_KeyInfo*) with a *DevAdmGetResp(S_KeyInfo)*. As per CA7-38: on page 1472 the DA only provides records for the requesting node as identified by the MADHeader:RequesterID[49]. There may be multiple packets per response.

Table 440 S_KeyInfo Attribute

CMsk[a] (bit)	Component	Access	Offset	Length	Description
0	IouGUID	RO	0	8 bytes	Node GUID of the IOU
1	Supv_Key	RO	64	8 bytes	Supervisor_Key that the client platform uses in any DevMgt MADs to that IOU.
-	ClientKeyCount	RO	640	2 bytes	The number of C_KeyInfo records that will be returned for a DevAdmGet(C_KeyInfo) query that contains this IocGUID and Supv_Key.
-	reserved		656	14 bytes	reserved - not used
2	SKey_Name	RO	128	64 bytes	A name associated with the Supervisor_Key

a. ComponentMask bit - This column indicates which bit in the ComponentMask is set when the initiator wants only records that match the corresponding component value in the query. Only components with CMsk bit numbers assigned can be specified.

Each attribute record returned by the DA specifies an IOU and the Supervisor_Key that the client platform uses to manage client pools on that IOU.

Normally, the DA should use the same Supervisor_Key on all IOUs. However, there are times when that is not possible, thus the client platform should be prepared to use multiple keys. It is also possible that a client platform has multiple Supervisor_Keys on the same IOU (i.e., can receive multiple S_KeyInfo attribute records specifying the same IOU with a different Supv_Key and different SKey_Name. The manager may wish to provide different Supv_Keys based upon the current function of the platform (e.g. Development and Test, versus using the system as a failover for production.) The SKey_Name allows the supervisor to select the appropriate Supv_Key from the set of all those returned.

A7.6.3.6 C_KEYINFO

The DA responds to *DevAdmGet(C_KeyInfo*) with a *DevAdmGet-Resp(C_KeyInfo)*. As per CA7-38: on page 1472 the DA only provides records for the requesting node as identified by the MADHeader:RequesterID[50]. There may be multiple packets per response.

49. When the IORM performs a LOGIN, the DA determines the IORM's Node GUID and assigns it a unique RequesterID, which the IORM specifies in the MAD Header of all its queries. The DA uses the RequesterID to authenticate the query and associate the request to a particular host platform.

Table 441 C_KeyInfo Attribute

CMsk[a] (bit)	Component	Access	Offset	Length	Description
0	IouGUID	RO	0	8 bytes	Node GUID of the IOU
1	Supv_Key	RO	64	8 bytes	Supervisor_Key that the client platform uses in any DevMgt MADs to that IOU.
2	Client_Key	RO	128	8 bytes	Client_Key that a client uses in any DevMgt MADs to that IOU.
	reserved	RO	192	8 bytes	reserved - not used
3	ClientKey_Name	RO	256	64 bytes	A name associated with the Client_Key

a. ComponentMask bit - This column indicates which bit in the ComponentMask is set when the initiator wants only records that match the corresponding component value in the query. Only components with CMsk bit numbers assigned can be specified.

Each attribute record returned by the DA specifies a Client_Key that the client platform can use for the specified IOU and the Supervisor_Key the platform' IORM uses to manage the ClientPoolRecord assigned that Client_Key.

Note that a Client_Key may be valid on more than one IOU, but it might not be valid on all IOUs. Thus, it is possible for the client platform to be given a different set of Clent_Keys for each IOU.

50. When the IORM performs a LOGIN, the DA determines the IORM's Node GUID and assigns it a unique RequesterID, which the IORM specifies in the MAD Header of all its queries. The DA uses the RequesterID to authenticate the query and associate the request to a particular host platform.

A7.6.3.7 REMOVALREQ

The DA uses the DevAdmReport(RemovalReq) to report the removal process to client platforms affected by the removal of an I/O module (see A7.5.8 "Graceful Hot Removal" on page 1464).

Table 442 RemovalReq Attribute

Component	Access	Offset	Length	Description
IouGUID	RO	0	8 bytes	The CA GUID of the IOU containing the I/O module being removed.
RemovePriority	RO	64	4 bits	See Table 443 on page 1499
Acknowledge	R/W	68	4 bits	In the DevAdmReportResp( ): 0=OK to remove 1=REJECT, critical resource, etc. 2=WAIT, need time to acquiesce operation In the DevAdmReport( ), this field is set to zero and ignored
WaitDelay	R/W	72	8 bits	Set to zero in the DevAdmReport( ) and when the IORM responds with an Acknowledge value of WAIT, the IORM sets this component to the number if milliseconds that the DA should wait before it resends the request. For each Report() the IORM may request more time.
reserved	RO	80	22 bytes	reserved
IocGUID1	RO	256	8 bytes	IOC GUID of an affected IOC.
IocGUID2	RO	320	8 bytes	GUID of an affected IOC (or zero if no more IOCs, means IocGUID3 through IocGUID16 are also zero)[a]
IocGUID3	RO	384	8 bytes	GUID of an affected IOC (or zero if no more IOCs)
IocGUID4	RO	448	8 bytes	GUID of an affected IOC (or zero if no more IOCs)
IocGUID5	RO	512	8 bytes	GUID of an affected IOC (or zero if no more IOCs)
IocGUID6	RO	576	8 bytes	GUID of an affected IOC (or zero if no more IOCs)
IocGUID7	RO	640	8 bytes	GUID of an affected IOC (or zero if no more IOCs)
IocGUID8	RO	704	8 bytes	GUID of an affected IOC (or zero if no more IOCs)
IocGUID9	RO	768	8 bytes	GUID of an affected IOC (or zero if no more IOCs)
IocGUID10	RO	832	8 bytes	GUID of an affected IOC (or zero if no more IOCs)
IocGUID11	RO	896	8 bytes	GUID of an affected IOC (or zero if no more IOCs)
IocGUID12	RO	960	8 bytes	GUID of an affected IOC (or zero if no more IOCs)
IocGUID13	RO	1024	8 bytes	GUID of an affected IOC (or zero if no more IOCs)
IocGUID14	RO	1088	8 bytes	GUID of an affected IOC (or zero if no more IOCs)

Table 442 RemovalReq Attribute (Continued)

Component	Access	Offset	Length	Description
IocGUID15	RO	1152	8 bytes	GUID of an affected IOC (or zero if no more IOCs)
IocGUID16	RO	1216	8 bytes	GUID of an affected IOC (or zero if no more IOCs)
reserved32	RO	1280	32 bytes	reserved

a. If an IocGUID is zero, it means there are no more IOCs in the list and the client platform may ignore the remaining IocGUIDs (i.e., ignore IocGUID components following the first zero IocGUID).

The DA sends a DevAdmReport(RemovalReq) to the IORM of each affected client platform setting the RemovePriority component to inform the IORM what options the IORM has to reject or delay the removal (see Table 443). The *Acknowledge* component identifies the willingness of the IORM to accept the removal. *Acknowledge* and *WaitDelay* are the only components that the IORM may modify when it returns the attribute in the DevAdmReportResp(RemovalReq).

The DA sets the *Acknowledge* component to zero and if *RemovePriority* permits, the IORM may change it in the response to reject the removal or request more time. See RemovePriority component above and Table 443 for IORM options in responding to the DevAdmReport(). If the IORM indicates WAIT, then the IORM also sets the *WaitDelay* component to tell the DA how long it should wait before asking again. If the client needs more than 255 msec, it sets *WaitDelay* to 0xFF. The DA continues to send a DevAdmReport() until the client indicates OK to remove (or reject).

Each report can identify up to 16 IOCs affected by the removal. When there are more than 16 the DA sends multiple reports. If an IocGUID is zero, the client platform may ignore the following IocGUIDs.

A RemovalReq can have one of the following priorities.

Table 443 Removal/Diagnostic/Reset Priority

Value	Description
0h	**Graceful** - if a client platform responds negatively to the notice, the DA cancels the removal/diagnostic/reset operation.
1h	**Forced** - Client Platforms are not allowed to reject the removal/diagnostic/reset, but may respond with WAIT status. The amount of time that the DA will permit a client platform to delay the action is a DA policy.
2h	**Immediate** - Client Platforms are not allowed to reject or delay the removal/diagnostic/reset - i.e., the DA ignores the Acknowledge component in the response. The amount of time that the DA attempts to contact a client platform before performing the action is a DA policy.
3h	**Critical** - The configuration manager issues the removal/diagnostic/reset command without waiting for acknowledgment from client platforms. The DA sends this notice to inform the client platforms of the action, but this report may come after the fact - i.e., the DA ignores the Acknowledge component in the response.

The architecture permits I/O module removal requests to be initiated by the IOU (see DevMgt Annex) or initiated via the Configuration Manager. The criteria for the Configuration Manager initiating a remove and how the DA determines the RemovePriority is a Configuration Manager policy outside the scope of this document. See A7.6.1.2.4 "Subscription Timeout:" on page 1475 for how the DA determines that a client platform is not responding. How the DA reacts to non-responding clients is a DA policy.

A7.6.3.8 DIAGNOTICE

The DA uses the DevAdmReport(DiagNotice) to report that a diagnostic session has been requested (see A7.5.9 "Diagnostics" on page 1468).

Table 444 DiagNotice Attribute

Component	Access	Offset (bits)	Length (bits)	Description
IouGUID	RO	0	64	The CA GUID of the IOU being tested.
DiagPriority	RO	64	4	See Table 443 on page 1499
Acknowledge	R/W	68	4	In the DevAdmReportResp( ): 0h=OK to perform diagnostics 1h=REJECT, critical resource, etc. 2h=WAIT, need time to acquiesce operation In the DevAdmReport( ), this field is set to zero and ignored

Table 444 DiagNotice Attribute (Continued)

Component	Access	Offset (bits)	Length (bits)	Description
WaitDelay	R/W	72	8 bits	Set to zero in the DevAdmReport( ) and when the IORM responds with an Acknowledge value of WAIT, the IORM sets this component to the number if milliseconds that the DA should wait before it resends the request. For each Report() the IORM may request more time.
reserved	RO	80	48	reserved
DiagSeverity	RO	128	4	Specifics the maximum severity level for a diagnostic test • 0x0 - Non-Intrusive - does not impact I/O performance • 0x1 - I/O performance reduced to 90% • 0x2 - I/O performance reduced to 80% • 0x3 - I/O performance reduced to 70% • 0x4 - I/O performance reduced to 60% • 0x5 - I/O performance reduced to 50% • 0x6 - I/O performance reduced to 40% • 0x7 - I/O performance reduced to 30% • 0x8 - I/O performance reduced to 20% • 0x9 - I/O performance reduced to 10% • 0xA - No I/O Service during diagnostics - connections remain • 0xF - No I/O Service - Connections terminated • other values reserved
DiagScope	RO	132	4	Specifies the scope of the objects that can be affected by diagnostic testing using this token: 0x0 - Entire IOU: The diagnostics can affect all I/O objects of all IOCs. 0x1 - reserved. 0x2 - I/O Controllers: The diagnostics can affect all I/O objects of the IOCs listed in the ObjectList component. 0x3 - Service Objects: The diagnostics can affect the I/O objects listed in the ObjectList component. other values reserved
reserved	RO	136	104	reserved
ObjectListCount	RO	240	16	Specifies the number of objects in ObjectList. When DiagScope is 'Entire IOU' then this value is zero. The max value allowed is 24 for IOCs and 12 for Service Objects.
ObjectList	RO	256	varies	Specifies the list of objects that can be impacted by a diagnostic testing. The content of this component is based on DiagScope and ObjectListCount: • Entire IOU: no data (this field size is zero) • I/O Controllers: List of IocGUIDs - thus the length of this component is 8n Bytes where n = ObjectList Count. • Service Objects: List of IocGUID+ServiceObjectID tuples - thus the length of this component is 16n Bytes where n = ObjectListCount

The DA uses this attribute to inform hosts that a diagnostic session has been requested.

The DA sends a DevAdmReport(DiagNotice) to the IORM of each affected client platform setting the DiagPriority component to inform the IORM what options the IORM has to reject or delay the diagnostic session (see Table 443). The Acknowledge component identifies the willingness of the IORM to accept the diagnostic session. *Acknowledge* and *WaitDelay* are the only components that the IORM may modify when it returns the attribute in the DevAdmReportResp(DiagNotice).

The DA sets the Acknowledge component to zero and the IORM may change it in the response to reject the session or request more time (if DiagPriority permits). See DiagPriority component above and Table 443 on page 1499 for IORM options in responding to the DevAdmReport(DiagNotice). If the IORM indicates WAIT, then the IORM also sets the *WaitDelay* component to tell the DA how long it should wait before asking again. If the client needs more than 255 msec, it sets *WaitDelay* to 0xFF. The DA continues to send a DevAdmReport() until the IORM indicates 'OK to perform diagnostics' (or reject).

Most of the information comes from the DevMgtSet(DiagSession). The DA only lists devices that are assigned[51] to the client platform and are listed in the DiagSession ObjectList. If the DiagSession lists I/O modules, then the DA has to convert the module list to a list of IOCs. If the DiagSession specifies more IOCs or Service Objects than will fit in the DiagNotice attribute, the DA sends multiple single packet Report(DiagNotice) MADs.

A7.6.3.9 RESETNOTICE

The DA uses the DevAdmReport(ResetNotice) to inform IORMs before it issues a reset to an IOU. The reset can be for a specific IOC or the entire IOU.

Table 445 ResetNotice Attribute

Component	Access	Offset (bits)	Length (bits)	Description
IouGUID	RO	0	64	The CA GUID of the IOU being reset or the IOU containing the IOC being reset.
ResetPriority	RO	64	4	See Table 443 on page 1499
Acknowledge	R/W	68	4	In the DevAdmReportResp( ): 0h=OK to perform Reset 1h=REJECT, critical resource, etc. 2h=WAIT, need time to acquiesce operation In the DevAdmReport( ), this field is set to zero and ignored

51. Assigned means the IOC or Service Object is listed in a Platform Pool Table record or in a Client Pool Table record for which the DA has provided the IORM a Supervisor_Key or Client_Key.

Table 445 ResetNotice Attribute (Continued)

Component	Access	Offset (bits)	Length (bits)	Description
WaitDelay	R/W	72	8 bits	Set to zero in the DevAdmReport( ) and if the IORM responds with an Acknowledge value of WAIT, the IORM sets this component to the number if milliseconds that the DA should wait before it resends the request. For each DevAdmReport() the IORM may request more time.
reserved	RO	80	48	reserved
ResetType	RO	128	4	Reset Type that will be specified in the DevMgtSet(Reset) to the IOU. • 0b - Graceful Reset - Shutdown and Reset. The IOU/IOC completes outstanding commands, closes files etc., then resets. • 1b - Immediate Reset - The IOU will perform the reset immediately.
ResetScope	RO	132	2	Scope of reset that will be specified in the DevMgtSet(Reset) to the IOU. 00 - default (IOU determines scope of reset) 01 - Reset only software 10 - Reset only hardware 11 - Reset both hardware and software
reserved	RO	136	104	reserved
IocGUID	RO	240	16	Specifies the IOC GUID of the IOC that will be reset. A value of zero indicates that the entire IOU will be reset.

The DA uses this attribute to inform hosts that it will issue a Reset to the specified IOU. The DA only sends the DevAdmReport(ResetNotice) to IORMs that will be affected[52] by the reset and have subscribed for the ResetNotice.

CA7-78: Before the DM sends a DevMgtSet(Reset) to an IOU, the DA shall issue a DevAdmReport(ResetNotice) to each client platform that satisfies the following requirements (a) the platform has been assigned a service object that will be affected by the reset, (b) the client platform subscribed for *Reset Notification,* and (c) a GID of the IOU matches the InformInfo:GID or the IOU has a LID that falls in the range of the InformInfo: LIDRangeBegin – LIDRangeEnd.

CA7-79: The DA shall not issue a DevAdmReport(ResetNotice) to a client platform unless at least one of the affected IOCs' service objects are listed in a PlatformPoolRecord for which the client platform has been given the Supervisor_Key or listed in a ClientPoolRecord for which the client platform has been given the Client_Key.

52. Affected means the IOC is listed in a Platform Pool Table record or in a Client Pool Table record for that client platform.

The DA sends a DevAdmReport(ResetNotice) to the IORM of each affected client platform setting the ResetPriority component to inform the IORM what options the IORM has to reject or delay the Reset (see Table 443). The Acknowledge component in the response identifies the willingness of the IORM to accept the Reset. *Acknowledge* and *WaitDelay* are the only components that the IORM may modify when it returns the attribute in the DevAdmReportResp(ResetNotice).

The DA sets the Acknowledge component to zero and the IORM may change it in the response to reject the Reset or request more time (if ResetPriority permits). See ResetPriority component above and Table 443 on page 1499 for IORM options in responding to the DevAdmReport(ResetNotice). If the IORM indicates WAIT, then the IORM also sets the *WaitDelay* component to tell the DA how long it should wait before asking again. If the client needs more than 255 msec, it sets *WaitDelay* to 0xFF. The DA continues to send a DevAdmReport() until the IORM indicates OK to Reset (or reject).

A7.7 COMPLIANCE

In order to claim compliance to Configuration Management a product shall meet all requirements specified in this section,

A7.7.1 COMPLIANCE CATEGORIES

This annex specifies two compliance categories: Configuration Manager and I/O Client. The Device Management Annex specifies the requirements for an I/O unit.

In order to claim compliance to Configuration Management a product shall meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support.

Table 446 Configuration Management Compliance Categories/Qualifiers

Category	Qualifiers	Description
CFM	none	Minimum requirements for a Configuration Manager
	PERS	Requirements for a manager that persistently saves client contexts such as subscriptions and diagnostic sessions across reset, restarts, and power cycles.
	FAILOVER	Requirements for a manager that supports graceful failover such that if the manager fails and a standby manager takes over, the new manager inherits the client context (subscriptions and diagnostic sessions) from the old manager.
Client Platform	none	Minimum requirements for client platforms

A7.7.2 CONFIGURATION MANAGER COMPLIANCE SUMMARY

In order to claim compliance to the InfiniBand Specification for the Compliance Category of Configuration Manager, a product shall provide a Device Manager and Device Administrator that meet all requirements specified in this section and in A7.7.4 "Common Management Requirements" on page 1506.

A7.7.3 I/O CLIENT COMPLIANCE SUMMARY

In order to claim compliance to the InfiniBand Specification for the Compliance Category of I/O Client, a product shall meet all requirements specified in this section and in A7.7.4 "Common Management Requirements" on page 1506.

A7.7.4 COMMON MANAGEMENT REQUIREMENTS

All managers and agents must be compliant with the Common MAD requirements specified is 20.14 and the following general management framework requirements from Chapter 13.

ANNEX A8: DEVICE MANAGEMENT

A8.1 INTRODUCTION

This annex in conjunction with Annex A7: Configuration Management defines the framework for managing I/O units and assigning I/O Unit resources to client platforms. This annex specifies the Device Management class and the requirements for an I/O unit to support Device Management (i.e., requirements for the Device Management Agent). The Configuration Management annex describes the overall configuration management framework, specifies the Device Administration Class, and specifies the requirements for a configuration management application that configures the I/O Unit (i.e., the Device Manager) and administers privileged configuration information to client platforms (i.e., the Device Administrator).

This annex specifies an enhanced Device Management class for managing I/O resources of an I/O unit and thus this annex specifies version 2 of the Device Management class which supersedes version 1 specified in Chapter 16 section 16.3.

The Device Management (DevMgt) class provides the means to manage an I/O unit and its I/O services. This class specifies formats for Device Management packets and the protocol for sending and receiving those packets between two IB endnodes, typically between an HCA of a client platform and a TCA of an I/O unit and between an HCA of the Device Manager and an I/O unit. To facilitate this, a managed I/O unit provides a DevMgt Agent that manages I/O unit attributes and configuration information.

I/O units are capable of providing I/O service to multiple clients. IB partitioning enforces isolation among systems sharing an InfiniBand fabric, but does not provide partitioning of the resources within a node. Device Management in conjunction with Device Administration (the service used to administer configuration information, see Annex A7: Configuration Management), and Communication Management (CM; see Chapter 12i) together extend the IB Architecture to provide enforceable assignment of I/O resources to multiple clients.

This annex describes the mechanism for a Device Manager to configure the IOU with the information necessary for the I/O unit to limit and control access to its services and resources on a client by client basis, thus enabling multiple platforms to use separate I/O devices located in the same I/O unit without interfering with one another.

This annex defines the wire-level interface to the DevMgt Agent and a wire-level protocol that provides a way to discover I/O service objects, configure those objects for particular clients, and provide information to clients necessary for those clients to access their service objects. The DevMgt class also provides a means for clients to invoke and manage diagnostics.

Support for Device Management is optional (i.e., not all nodes contain a DevMgt agent). A node that contains a DevMgt agent is considered a managed I/O unit. This annex addresses managed I/O units and the entities that access them.

A8.1.1 GLOSSARY

Annex A7: Configuration Management defines additional glossary terms used in this annex.

A8.1.2 COMPLIANCE

This annex specifies a new Compliance Category. See Chapter 20: Volume 1 Compliance Summary on page 1072 for explanation of compliance categories and qualifiers. The new category is DevMgt Agent.

The Compliance Qualifiers for DevMgt Agent are:

- V1: Backward compatibility with class version 1.
- HOT: Hot plug/removal of I/O modules

Section A8.9 Compliance on page 1625 provides a summary of compliance statements for a DevMgt Agent.

A8.1.3 GOALS AND OBJECTIVES

This annex revises Device Management as follows:

1) Enhanced Device Management for enterprise class I/O
   - Separates definition of I/O subassembly module from I/O controller and allows multiple IOCs per subassembly slot.
   - Adds support for hot removal of I/O subassemblies
   - Allows I/O units and I/O controllers to support multiple protocols. These protocols include management protocols as well as I/O protocols.
   - Provides a means to identify a range of protocol versions supported.
   - Object oriented - adds the notion of service objects and provides for their identification, identification of their properties, protocols and protocol properties, and the ServiceIDs for accessing service objects.

- Provides clearer separation between I/O objects, their protocol, and the information needed to access the I/O object.
- Adds Vendor ID and Description information to IOUnitInfo.
- Removes fields that were not applicable.

2) Aligns DevMgt with configuration management.
   - Provides enforceable control of resources within an I/O unit where multiple clients share the same I/O unit.
   - Supports both passive and active configuration management models
3) Supports both simple and elaborate I/O Unit implementations

The bounds and limitations of this annex are:

1) Managing & configuring I/O devices behind the IOC, such as LUN mapping and other I/O protocol-specific attributes are the responsibility of I/O management protocols and are outside the scope of Device Management. However, Device Management does provide the means to advertise which I/O management protocols the IOU/IOC supports.
2) DevMgt does not mandate how I/O devices are mapped to I/O service objects. The relationship can be physical, logical, or virtual. For example:
   a) Secondary I/O ports could be mapped as I/O Service Objects; for this case, the client uses one channel (or one set of channels) to access all I/O devices accessible through that port.
   b) Each I/O device attached to the secondary fabric could be mapped as an I/O Service Object; for this case, the client uses one channel (or a set of channels) for each I/O device - that is each I/O device has its own QP (or set of QPs).
   c) There could be virtual and logical service objects (one to many, many to one, and many to many mappings between I/O service objects and I/O devices).

A8.2 OVERVIEW

The primary goals of Device Management (DevMgt) are to provide information about an I/O Unit (IOU) and its service objects and to provide enforceable access of service objects by client platforms. A service object is a port through which a client can access or manage I/O devices. Thus, each service object has its own set of QPs.

A8.2.1 USAGE MODEL

Device Management is used by a number of different entities. These include the Device Manager (DM), I/O resource manager (IORM), I/O client,

and I/O management application (see Figure 307 ”Device Management Usage Model” on page 1510 and Figure 309 ”I/O Components and Relationships” on page 1517).

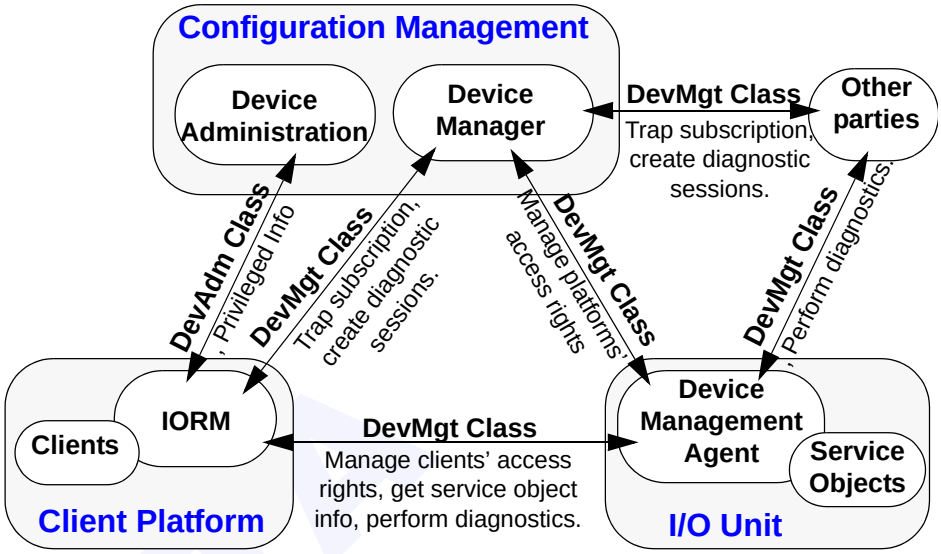


Figure 307 Device Management Usage Model

Device Management provides the means for a DM to provision IOU resources and allocate them on a per client platform basis. This provides the client platform’s IORM the ability to subdivide IOU resources allocated to that platform to individual clients based on OS policy.

A8.2.1.1 DEVICE MANAGER (DM)

The Device Management class provides mechanisms (methods and attributes) to enable a DM to set and retrieve I/O device specific information from I/O units.

A DM is a management entity that sets Device Management attributes of an I/O unit (i.e., sends a DevMgtSet() to a DevMgt agent). Only the DM has access to all of the IOU’s attributes. To enforce this, Device Management class provides a Device Manager’s key (*Manager_Key*) that restricts DevMgtSet() operations for certain attributes to authorized managers, i.e., those who know the Manager_Key.

The manager key allows the DM to create *Platform Resource Pools* and *Client Resource Pools*. These pools are referred to as Platform Pool Table Records and Client Pool Table Records.The DM uses a Platform Pool Record to specify which resources a particular platform may use and the platform’s IORM uses Client Pool Records to assign those resources to individual clients.

Each *Platform Resource Pool (i.e., Platform Pool Record)* is identified by a unique Supervisor_Key and each *Client Resource Pool (i.e., Client Pool Record) is* identified by a unique Client_Key. The manager associates each client pool to a specific Supervisor_Key. The manager provides the Supervisor_Key to the specific client platform (i.e., the platform's IORM). This permits an IORM with the correct Supervisor_Key to assign IOU resources specified in that Platform Pool Record to client resource pools. When the IORM passes a Client_key to one of its clients, that key enables the client to access the I/O resources specified by that client pool record.

Management applications and I/O clients also access IOU information via DevMgt class MADs (i.e., read attributes) but the DM and the IORM are the only ones that may program the IOU. This programming specifies which service object a client may see and use.

It is not always necessary to have a DM. "*Passive configuration management*" is the term used when a fabric is designed to function without a DM. Passive configuration management is typical in a subnet where each IOU is dedicated to a single client platform (i.e., each IOU configured for a single partition containing one I/O client platform) or where all clients in that partition have equal access to all of the IOU's resources. Except for the most trusted environments, a DM is needed when IOUs are divided among multiple clients.

In general, a Device Manager:

- Uses DevMgt to learn what I/O service objects are available, so they can be assigned to client platforms. DevMgt supplies the DM with information about service objects, such as number clients the service object supports as well as the number and type of QPs that an I/O object uses to communicate with a client.
- Uses DevMgt to configure each IOU as to which client platforms are authorized to see and use which service objects.
- Configure IOUs to send DevMgt traps to the DM. The DM forwards those traps to nodes that have subscribed for them.
- Controls Diagnostic Sessions on each IOU, including the ability to reset the IOU and I/O controllers.

In contrast, for passive device management, any client platform that can access an I/O unit (i.e., is assigned a common P_Key) has complete access to all of the IOU's Device Management information and I/O resources.

An IOU defaults to passive configuration management mode, but is placed in active configuration management mode by virtue of a DM setting the IOU's Manager_Key and programming the IOU's pool tables.

Client Platforms and other managers use Device Management to subscribe to the DM to receive reports when IOUs generate Device Management Traps and to establish (via the DM) a Diagnostic Session so it can invoke diagnostics on an IOU or IOC. Note that the diagnostic application requests a diagnostic session via the DM, but preforms diagnostics directly with the IOU.

The Device Management framework supports the concept of multiple DMs but for an IOU, there is only one logical DM. That is, the one that knows the IOU's Manager's key. Thus, an IOU does not distinguish between DMs and accepts any MAD containing the proper Manager_Key as coming from an authorized DM. The DM programs each of the IOU's ports where to send traps. Even though each port's ClassPortInfo can specify a different trap destination, a valid TrapRepress() received on any port represses that trap for all ports.

A8.2.1.2 I/O RESOURCE MANAGER

An I/O Resource Manager (IORM) is the component of an operating system that supervises I/O resources for that platform. An IORM uses DevAdm class (see Configuration Management Annex) to query the Device Administrator associated with the DM to learn the platform's Supervisor Key and list of IOUs. It then queries the IOU using DevMgt to discover which I/O resources (i.e., I/O service objects) are available to that platform and characteristics about them that enables the IORM to load an I/O driver that executes the appropriate I/O protocol (see "I/O Drivers" in Annex A1).

An IOU uses the Supervisor_Key to distinguish between IORMs and accepts any MAD containing a valid Supervisor_Key as coming from that client platform.

When an IORM sends DevMgtGet()s to an IOU, it provides its Supervisor_Key, and the information that the IOU returns is dependent on how the DM configured the platform pool record that has that Supervisor_Key.

The IORM, as the node's supervisor, can discover how many Client Pool Records are assigned to the client platform, and can configure them specifying which service objects and the resources (such as the number of QPs) that a client using that Client_Key is permitted to consume.

A8.2.1.3 I/O CLIENT

An I/O client is the entity that actually communicates with an I/O object using an I/O protocol. The IOU has Client Pool Records to manage what a client can see and use. Each Client Pool Record has a unique Client_Key and is assigned to a particular platform. The platform's IORM configures Client Pool Records to specify client access rights that deter-

mine which service objects a client using that Client_Key can see and access, and then makes the Client_Key known to the client.

An IOU uses the Client_Key to distinguish between clients and accepts any MAD containing a valid Client_Key as coming from that client.

When an I/O client sends DevMgtGet()s to an IOU, it provides its Client_Key, and the information that the IOU returns is dependent on how the IORM configured the Client Pool Record that has that Client_Key.

The I/O client uses DevMgt to find connection parameters, such as the ServiceID, that it needs to establish communication (i.e., establish RC, UC, RD, and/or UD channels) with the service object. Once the I/O client establishes communications, it uses those channels to exchange I/O protocol packets with the I/O service object.

The client uses its Client_Key when setting up channels (e.g., passing its Client_Key in CM MADs or in an I/O protocol specific login sequence after the channel is established). The CM, DevMgt Agent, and Service Objects work together to enforce the I/O access rights of a client by searching for the Client_Key in the Client Pool Table. If the Client_Key is found, then that record specifies the resources for the client. The CM denies the connection (e.g., CM:REJ) if the Client_Key is not found, the client is not authorized for the service object, or making the connection would cause a maximum allocation level to be exceeded.

A8.2.1.4 I/O MANAGEMENT APPLICATION

An I/O management application is an entity that manages I/O properties of an I/O unit (such as to create, configure, destroy I/O service objects). The system administrator might wish to run such an application from its own machine rather than from the machine that is actually using the I/O services.

In addition to the IB management framework (which provides for vendor defined management classes using MADs), Device Management architecture provides for management protocols that don't use Vendor MADS.

Device Management architecture supports these additional management protocols because some IOCs, such as a RAID controller, may require extended management, where an I/O management application communicates with a management service object using a specific I/O management protocol to configure I/O device specific operation.

Device Management provides the means to advertise if an IOU or IOC supports a particular I/O management protocol. For each supported I/O management protocol, there will be a service object that identifies the I/O management protocol and provides connection information, such as the

ServiceID, that the management application uses to establish communications with that service object. The management application, as a client, is provided with a Client_Key and can only send DevMgtGet()s to the IOU's DevMgt Agent (that is, DevMgtSet()s are not permitted). The information that the IOU returns is dependent on how the Client Pool Record that has that Client_Key has been configured.

In addition, the IOU uses the Client_Key to validate that the client is authorized to access that management service object, such as requiring the management application to pass the key when creating a channel (e.g., in the CM message) or in a management protocol specific login sequence after the channel is created. Thus, the DM controls which nodes may perform I/O management using pool tables just like it does for controlling I/O service objects.

A8.2.2 I/O UNIT MODEL

This section explains the I/O unit service model on which Device Management is based.

An I/O unit contains one or more I/O controllers and associated service objects.

- An *I/O Controller* (IOC) is a circuit and/or process of an IOU that provides I/O service. That is, an IOC provides one or more I/O service objects.
- A *Service Object* refers to an instance of service that is addressed by its QPs (i.e., a client uses a different channel to communicate with each service object).
- I/O devices, such as disk drives, can be presented as individual service objects (i.e., each with its own QP) or as protocol objects behind a service object.
- Service objects that perform I/O are referred to as *I/O service objects*.
- Management agents on the IOU used to configure I/O service objects are also represented as service objects. They are referred to as *I/O management service objects*.

Although this annex uses language implying that an IOU "contains" IOCs, this annex does not specify packaging requirements. Thus, how IOCs are connected to an IOU is an implementation detail outside the scope of the architecture. Figure 308 Model for an I/O Unit on page 1515 provides the architectural and connection model for an IOU, consisting of a CA and one or more IOCs.

.

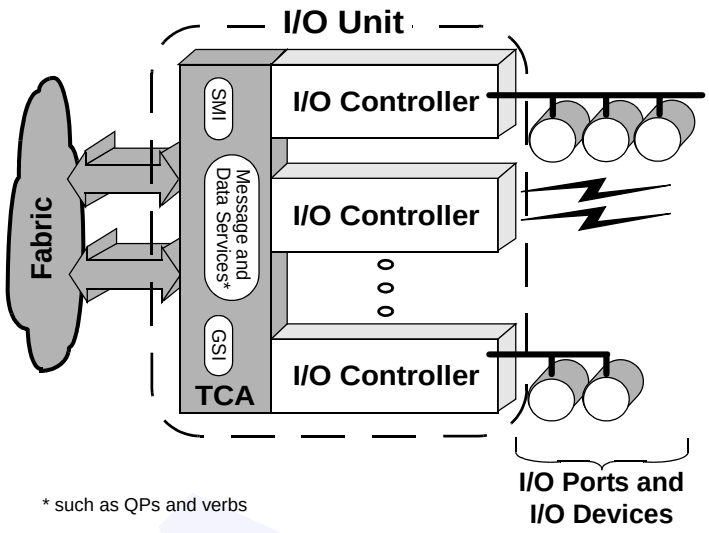


Figure 308 Model for an I/O Unit

An IOU attaches to the fabric via a CA. For I/O operation, the CA receives packets from the fabric and delivers complete, valid IBA messages to service objects, and vice-versa. The service object is then responsible for executing I/O requests contained in those messages, such as network sends and receives or disk reads and writes over a device-specific interface such as Ethernet, Parallel SCSI, Fibre Channel, or a proprietary interconnect.

The Device Management Agent is an IB management agent (as defined in Chapter 13) that resides behind the CA (i.e., behind the GSI) and interfaces with the IOU and its IOCs to manage I/O specific information. This annex focuses on the infrastructure, related methods, data formats, and attributes to support IOU/IOC management over the fabric. It uses the management framework specified in Chapter 13 to send and receive DevMgt packets to/from the DevMgt agent, but does not define the implementation-specific mechanisms necessary to translate MADs into a format that the IOU and IOCs understand, nor does it define how that data is delivered and retrieved from the IOU and IOCs.

The InfiniBand architecture is based on message passing. For an IOU and its service objects, the messages generally fall into three categories: fabric configuration, unit management/configuration, and I/O transaction:

1) Fabric configuration messages are processed by the Subnet Management Agent (SMA) and are defined in Chapter 14: Subnet Management on page 794. They are not discussed in this Annex.

2) Messages specific to configuring and managing I/O properties of an I/O unit fall into two categories:

   a) Generic DevMgt messages used to describe the IOU, IOC, and service objects, to control access to DevMgt information, and to perform diagnostics. These messages are received through the General Services Interface (GSI) and are described in this annex.

   b) Implementation-specific I/O management messages used to create, configure, and destroy service objects. While implementation-specific management could use vendor class MADs and/or private vendor attributes, DevMgt also supports the notion of implementation-specific management outside the IB management framework. These messages are sent from I/O management applications to I/O management service objects of the IOU/IOC. The definition of these messages are outside the scope of this document. However, DevMgt provides the means to identify I/O management objects, the management protocol, and obtain information that permits a management application to create channels to those I/O management objects. In this annex, I/O management objects mean those implemented outside the IB management framework.

3) I/O transaction messages used for I/O transactions (such as disk reads and writes). I/O transaction messages include those messages used by an I/O client to request I/O services from a service object, messages containing user or application data, and messages used by the service object to provide a completion notification (ending status) to the requester. Definition of these messages are also outside the scope of this document. However, DevMgt provides the means to identify I/O service objects, their I/O protocol, and information that permits an I/O client to create channels to the service objects.

A8.2.3 DEVICE MANAGEMENT MODEL

Figure 309 I/O Components and Relationships on page 1517 illustrates the relationships between the various entities and elements involved in I/O operations and management.

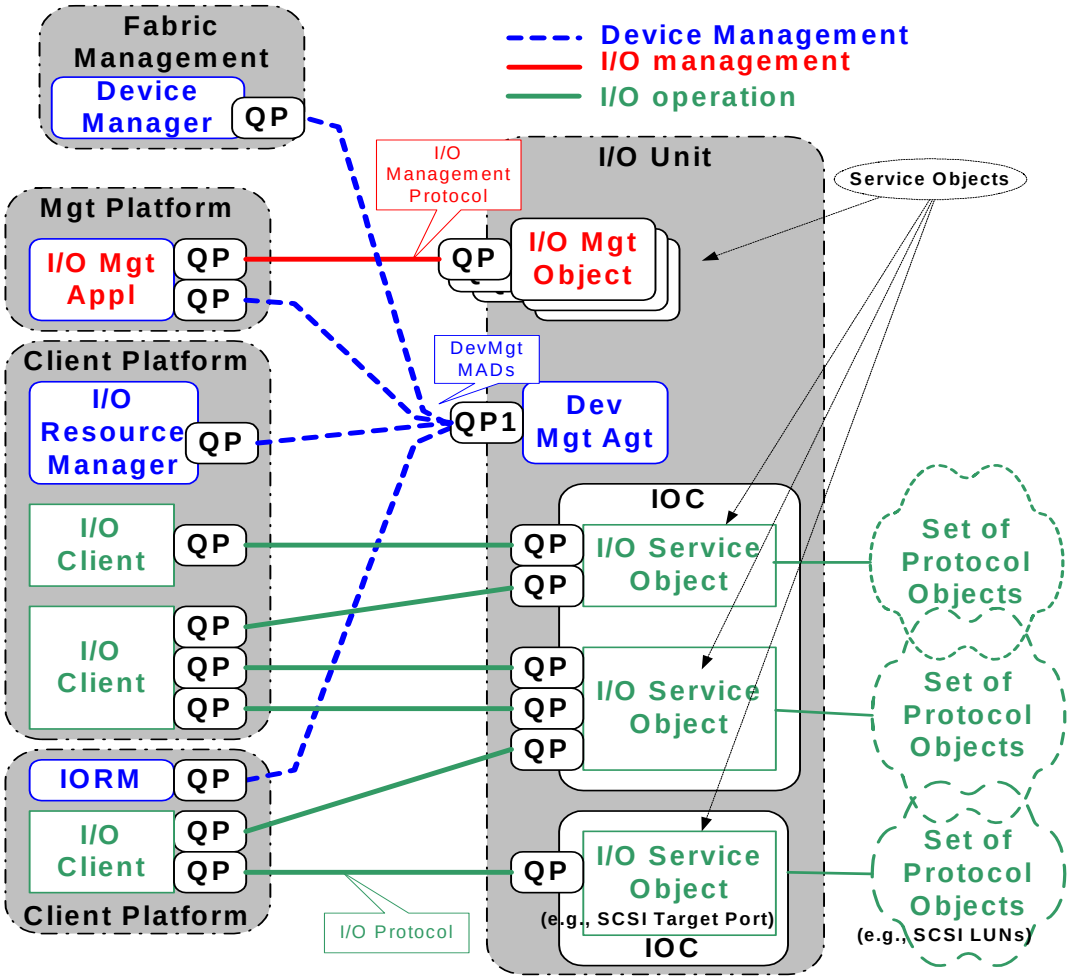


Figure 309 I/O Components and Relationships

Figure 309 illustrates that the Device Manager, I/O management applications, and client platforms access the DevMgt Agent via DevMgt MADs (blue dashed line). I/O management applications create channels to I/O management service objects (red line) that carry I/O management protocol messages and I/O clients create channels to I/O service objects (green heavy line) that carry I/O protocol messages.

Typically, an I/O client establishes a single channel with a service object. However, Device Management does not prohibit using multiple channels.

The set of protocol objects shown to the left behind an I/O service object illustrates that further division of services beyond the I/O service object is a function of the I/O protocol. An example of an I/O service object and its

protocol objects is a SCSI target port and its logical units. The difference between service objects and protocol objects are that service objects are discovered via Device Management (blue dashed line) and accessed via their QPs (green heavy line), while protocol objects are discovered and addressed via the I/O protocol over the channels provided by those QPs (green heavy lines). Thus, the service object is a port through which a client can access protocol objects.

Device Management encompasses several categories of management.

- Authority - provides for authentication, protection, and configuration of the I/O unit by a DM.
- Device Information - Provides the Device Manager, management applications, and I/O clients with information about the IOU and the services provided by each IOC. This includes information about the I/O protocols and management protocols supported by the IOU and each IOC.
- Device Assignment - provides the means for a DM to configure client access rights to I/O objects.
- Device Diagnostics - provides the means to control the execution of diagnostics on various components of an I/O unit and report their results.
- Physical hot plug management - DevMgt supports the concept of physical subassemblies (I/O modules) which house IOCs and allow them to be inserted and removed from the I/O unit. The architecture supports up to 256 I/O module slots per IOU with any number of IOCs per slot and any number of service objects per IOC.

A8.2.3.1 AUTHORITY

The DevMgt class provides for both passive and active device management models. Passive management refers to only using partitions to limit access to an IOU, while clients in that partition share access to all of the IOU's resources. Thus, a DM is not necessary for passive management. Passive management is generally appropriate when an IOU is assigned to a single client platform (e.g., all ports of the IOU are configured for a single partition and there is only one client platform performing I/O that is a member of that partition). It is also appropriate when it is acceptable for all clients to have access rights to all of the IOU's service objects.

Active management refers to the presence of a DM, which takes charge of the IOU and configures the IOU as to which clients may access which service objects. Active device management is valuable where multiple client platforms need access to a particular IOU, but not the same service objects. Active device management can prevent one platform from disturbing or impacting the ability of other clients to perform I/O by restricting an I/O object so it can be accessed only by critical clients.

To facilitate active management, DevMgt class provides for a Device Manager's Key that can be set by the Device Manager. Only a manager with the correct key is able to set critical DevMgt attributes. This includes the ability to set trap info in ClassPortInfo, create /modify Supervisor_Keys and Client_Keys, assign resources to platform pools, and create diagnostic sessions.

The default settings for a DevMgt agent enable passive management. This means that the IOU is not protected (i.e. Manager_Key=0) and any node in the IOU's partitions can access any of the IOU's service objects. Configuring an IOU for active management implies setting the agent's Manager_Key to a non-zero value. As long as management sets the Manager_Key before programming the IOU's I/O partitions, client platforms will only see IOUs that are actively managed. Thus, when an IOU comes on-line, the subnet manager should insure that the IOU's DevMgt Agent is configured before assigning P_Keys to the IOU.

A8.2.3.2 DEVICE INFORMATION

A primary purpose of DevMgt is to enable communication between clients and their service objects and prevent a client from accessing a service object for which it is not configured. This allows enterprise class IOUs to support multiple clients and provides a standard architected method of enforcing which clients may access which I/O Objects.

- I/O management applications can discover I/O management objects. A management application uses DevMgt to discover if the IOU/IOC supports its management protocol, and if so, uses DevMgt information to connect to the I/O management object. Via that channel, the management application performs I/O-specific and device-specific management functions such as creating, destroying, or configuring I/O service objects. Thus, DevMgt provides the means for the IOU and IOC to advertise the presence of I/O management objects, identify the I/O management protocol, and provides information necessary for an I/O management application to access the appropriate management object.
- IORMs and I/O clients discover I/O service objects and associated properties. DevMgt provides the means to discover I/O service objects, identify the I/O protocols they support, and retrieve the information (such as Service ID) necessary for the client to access the object.

DevMgt uses a hierarchy of information in managing device information as illustrated in Figure 310.

The IOU is the parent object that has one or more IOCs. Each IOC can support multiple protocols and can have multiple service objects that are accessible by one or more of these protocols.

DevMgt defines a number of attributes that describe the IOU, the IOCs, protocols, service objects and the channels for accessing the service objects. In the figure, attributes are shown next to its associated object illustrating the relationship.

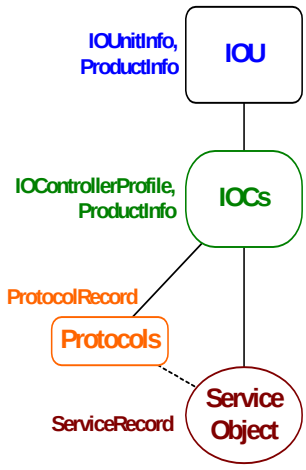


Figure 310 Data Hierarchy

- There is an IOUnitInfo attribute that provides information about the IOU in general and indicates the number of I/O modules (i.e., number and status of slots in the IOU). The IOU can provide an optional ProductInfo attribute that supplies additional product specific information about the IOU.
- The IOControllerProfile attribute provides information about an IOC. There is one attribute record for each IOC. An IOC is identified by its IocGUID. The IOC can provide an optional ProductInfo attribute that supplies additional product specific information about the IOC.
- The ProtocolRecord attribute specifies the characteristics of an I/O service protocol or an I/O management protocol. For each protocol that an IOC supports, there is one ProtocolRecord attribute. A protocol can be identified by its protocol name or its protocol ID.
- The ServiceRecord attribute provides the ServiceID a client needs for communicating with a service object via a specific protocol. A service object is uniquely identified by the combination of its IocGUID plus its ServiceObjectID. There is at least one ServiceRecord attribute record for each service object and can be more if the service object supports multiple protocols.

A8.2.3.3 DEVICE ASSIGNMENT

Device Management provides the means for a DM to configure the IOU as to which platforms may access which service objects and limit certain IOU resources (such as IOU QPs) that the platforms may consume. It also allows the platform's supervisor to allocate those resources to particular clients.

The IOU asserts access control at two levels. One is the ability to restrict certain Device Management information to specified supervisors and clients. The other is to restrict access to the service to only those clients that have been permitted access. That is, allowing a client to make a connection with the service object only if the client resource pool with that client's Client_Key specifies the service object and the client has not exceeded its IOU-QP allotment.

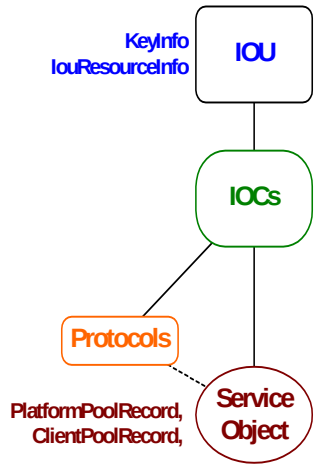


Figure 311 Device Assignment Attributes

A8.2.3.3.1 RESOURCE ALLOCATION

The IOU maintains a table of platform pool records. Each *PlatformPoolRecord* specifies the set of service objects that the platform may use and the resources (such as number of IOU QPs) that the platform may consume. Each record is identified by a unique Supervisor_Key. Only the DM can modify a PlatformPoolRecord. Once the platform knows its Supervisor_Key, it can read its PlatformPoolRecord to learn of its resources.

The IOU also maintains a table of client pool records. Each *ClientPoolRecord* specifies the set of service objects that the client may use and the resources (such as number of IOU QPs) that the client may consume. Each record is identified by a unique Client_Key. The record also specifies the Supervisor_Key of the platform that may modify the client pool. Only the DM can create a ClientPoolRecord, change the Client_Key and change the Supervisor_Key. The supervisor may modify a ClientPoolRecord, but can only assign resources allocated to the platform by the platform's PlatformPoolRecord.

When a client attempts to make a connection with a service object, the DevMgt agent only permits the connection if the ClientPoolRecord indicates that the client is authorized to access the service object and the client has not exceeded its authorized resources.

A Client learns its Client_Key from its IORM. An IORM learns its Supervisor_Key from the DA associated with the DM. For passive man-

agement (when there is no DM), the platform uses the default Supervisor_Key value of zero. Note that a node that boots different operating systems or boots up for different purposes might be given a different Supervisor_Key for each OS or each purpose, such that the Supervisor_Key controls which I/O resources the node may access.

The DevMgt Agent uses the Access_Key in the MAD header to determine which resources that the requestor is allowed to access and thus what information that the DevMgt Agent returns.

A8.2.3.3.2 QP ALLOCATION

The concept is that the DM assigns resources to platforms (such as reserving a number of IOU QPs for a platform and limiting the maximum number of IOU QPs the platform may use) and then the platform's supervisor assigns those resources to clients (within the bounds of the resources the DM assigned to the platform). See A8.4 Resource Allocation Framework on page 1606 for details.

The IOU's QPs are represented as a general pool of QPs which can be reserved for client platforms or available on a 'first come' basis. The DM specifies a minimum (Platform QPmin) and a maximum (Platform QPmax) number of IOU QPs for each client platform as illustrated in Figure 318 Allocating Resource Pools on page 1607. QPmin represents the number of QPs reserved for that platform. When the DM increases a platform's QPmin, the number of Free QPs in the general pool decreases and the number reserved increase. Thus, a portion of the general pool becomes reserved for specific platforms. FREE QPs are available to platforms on a first come first served basis as long as the platform has not reached its maximum limit.

A supervisor allocates QPs from its platform pool to client pools by specifying a minimum (Client QPmin) and a maximum (Client QPmax) number of IOU QPs for each client. The Client QPmin specifies QPs reserved for that client and the total number of QPs reserved for its clients cannot exceed the Platform QPmin (i.e., the number of QPs reserved for the platform).

As a client makes connections, the IOU uses QPs from the appropriate client pool. When a client has consumed all of the reserved QPs in its client pool (i.e., ClientQPmin), it may continue to consume additional QPs only if there are excess QPs available and the client pool has not reached its maximum limit. Thus, if the platform has un-allocated QPs (i.e., sum of clients QPmin is less than the platform's QPmin), then clients consume the platform's un-allocated QPs until they are exhausted. Once all the platform's un-allocated QPs are used, the clients may continue to consume available QPs from the general pool as long as the client's pool and the platform's pool have not reached their maximum limits. This means

that the number of reserved QPs plus additional QPs is less than QPmax both at a client level and at the platform level.

A8.2.3.3.3 SHARED POOLS

The architecture supports the notion of shared "*resource pools*". This concept is useful when there are more client platforms than PlatformPoolRecords. The DM can configure a platform pool as shared and lock the associated ClientPoolRecords. This permits the same Supervisor_Key to be used by multiple platforms and prohibits those platforms from modifying the ClientPoolRecords. Those supervisors simply provide the Client_Key to one of its clients and all of the clients with that Client_Key may access the service object(s). The DM should create an individual ClientPoolRecord for each service object of a shared pool, so that the supervisors can control which of its clients use which service objects.

A8.2.3.3.4 CLIENT IDENTIFICATION AND DEFAULT POOLS

DevMgt MADs contain an Access_Key component, where the DM, supervisor, or client specifies its Manager_Key, Supervisor_Key, or Client_Key respectively. The DevMgt Agent uses the Access_Key to validate the requestor.

Clients also use Communication Management to establish communication and thus consume QPs. However, CM MADs do not contain an equivalent key component. Most I/O protocols require some form of validation, where client information is passed to the service object for validation. Some protocols pass the equivalent of the Client_Key in the PrivateData field of the CM MADs. Others pass the equivalent of the Client_Key over the connection, but before the client is allowed to access the service object (e.g., some form of Login). Because I/O protocols already have architected ways of passing this type information, this annex does not specify how the information is passed.

Since not all I/O protocols validate who the client is, there is a CMValidation bit in the ServiceRecord that identifies if the ServiceObject validates Client_Key when the client creates a connection (see A8.3.3.7.1 CM Validatation on page 1567 for more details).

- For the case where the service object does have the ability to validate, the service object uses the key passed to it by the client to determine which Client Pool to use in order to determine if the connection should be made.
- For the case where the service object does not have the ability to validate, it has no way to know the client's client_Key, and thus the DM needs to specify which client pool is used for connections to that service object. To do this the DM can set the DefaultPool bit in a ClientPoolRecord to indicate that record defines a special "Default Pool" for the service objects listed in that record's ServiceObjList.

Typically, the DM creates a default pool for each service object that does not set the CMValidation bit. That Client_Key is not given to supervisors nor clients. Rather the DM includes the service object in the PlatformPool-Record for each platform that is authorized to use that service object. The only difference is that when clients connect to the service object, they do not consume QPs from their own client pool, but rather the service object consumes QPs from the DefaultPool (if available).

When the CMValidation bit is not set, Device Management still provides protection against unauthorized access because the platform (and its clients) cannot access the service record unless the service object is listed in the PlatformPoolRecord. Thus, the ServiceID can only be learned by an authorized client and since the client needs the ServiceID to make the connection, connections can only be made by authorized clients.

The CMValidation bit not being set only means that because the IOU cannot identify individual clients when making connections, those service objects use a default client pool, which limits the number of QPs all clients together may consume. Thus, the DM is not able to prevent one client from hogging all of those QPs. This does not effect QPs reserved for clients using services objects that do validate.

A8.2.3.3.5 PASSIVE MANAGEMENT

For passive management, the IOU defaults to a single shared Platform Pool record with Supervisor_Key=0, that includes all service objects, has a QPmin of 0, and a QPmax equal to or greater than the number of QPs in the general pool. This gives each platform full access to all of the IOU's information and service objects. The IOU also provides a set of Client Pools, one for each service object. Each Client Pools has a unique Client_Key arbitrarily chosen by the DevMgt agent.

A8.2.3.4 PHYSICAL MANAGEMENT

A8.2.3.4.1 MODULAR SUBASSEMBLIES

Because I/O units might be built with modular subassemblies (I/O modules) that can be added or removed, Device Management provides the means to identify which IOCs are associated with an I/O module and the means to gracefully remove or replace I/O modules.

Each I/O module houses one or more IOCs. Removal of an I/O module means removal of all of the associated IOCs. Before an I/O module can be removed, I/O clients using those IOCs need to cease I/O operations. Device Management provides the means to identify which IOCs are associated with each I/O module, defines a trap for the IOU to request the removal of an I/O module, the means for the DM to signal that it is OK to remove module power and remove the module.

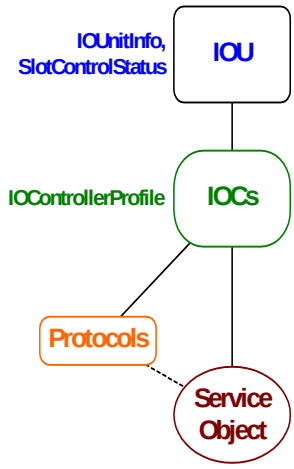


Figure 312 Physical Management Attributes

Each I/O module is represented by a slot number and the IOControllerProfile for each IOC identifies the slot to which the IOC is associated. The IOUnitInfo attribute specifies the number of slots and indicates if each slot is populated or not. The IOControllerProfile indicates with which slot the IOC is associated so that if an I/O module needs to be removed, powered down, or reset, the DM can determine which IOCs will be affected.

A8.2.3.4.2 GRACEFUL HOT REMOVAL

The IOU provides a SlotControlStatus attribute (one record for each slot) which provides the DM with the status of each slot and the means for the DM to control that I/O module's removal state.

Removable I/O modules contain a Remove switch, a Status LED that is normally ON when the module is in use, and an Attention LED. When an operator engages the Remove switch, the DevMgt Agent starts blinking the Status LED and sends a trap to the DM. When the DM receives the trap, the DA associated with the DM notifies the affected clients (see A7.5.8: Graceful Hot Removal in Annex A7:). When the clients notify the device manger that they have ceased their I/O operation, the DM signals the IOU that it is ok to remove the I/O module and the IOU turns off the Status LED as an indication that the operator can safely remove the module. The DM can cancel the removal process, in which case the IOU turns the Status LED ON instead of OFF.

The DM can initiate the removal process, causing the Status LED to start blinking without having an operator engage the removal switch - the rest of the process remains the same.

The DM can also control the Attention LED causing it to turn on, off, or blink.

A8.2.3.4.3 I/O MODULE EXAMPLES

Figure 313 provides an example of an IOU containing 6 permanent IOCs in one non-plugable I/O module. The IOUnitInfo attribute indicates one slot and 6 IOCs. Thus, there will be 6 IOControllerProfile records, each indicating the IOC is in slot 0. Because there are no removable slots, there are no SlotControlStatus attribute records.

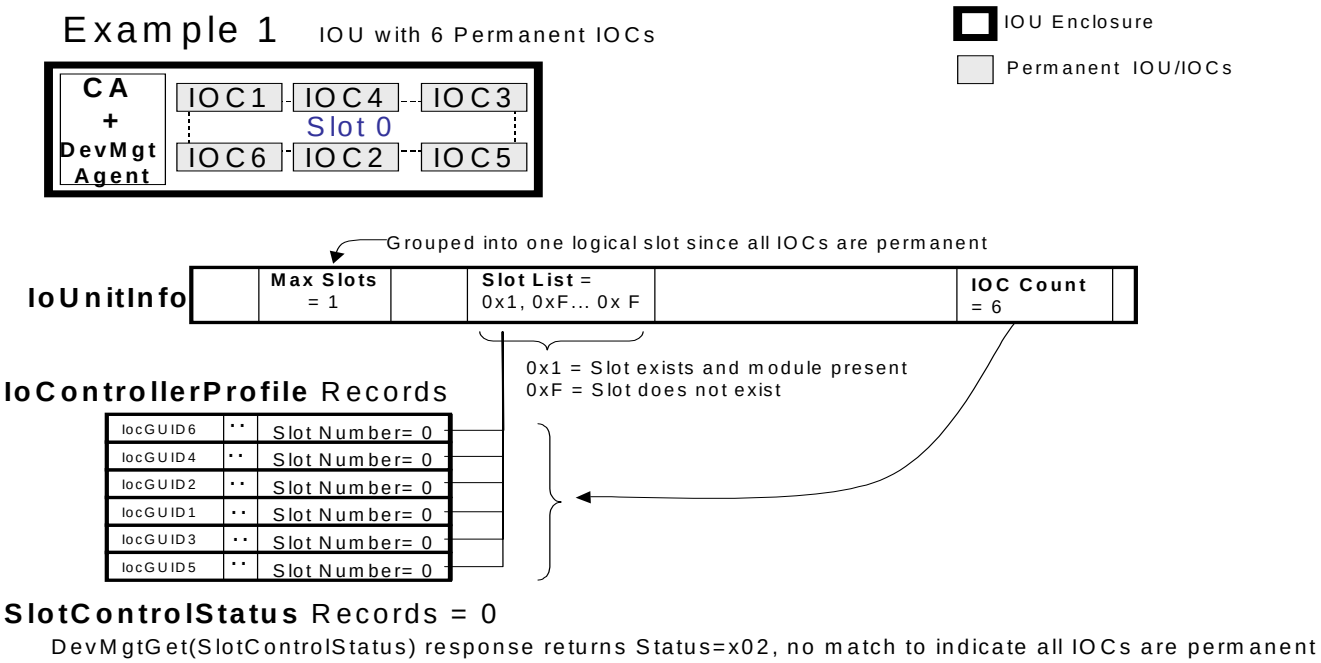


Figure 313 Example of IOU with Permanent IOCs

Figure 314 provides an example of an IOU containing 6 IOCs distributed between 3 I/O modules (1 permanent and 2 removable). The IOUnitInfo attribute indicates 3 slots and 6 IOCs. Thus, there will be 6 IOController-Profile records, each indicating its associated slot. Because there are 2 removable slots, there are 2 SlotControlStatus attribute records. The DM writes the SlotControlStatus records to control the hot plug status of the removable modules.

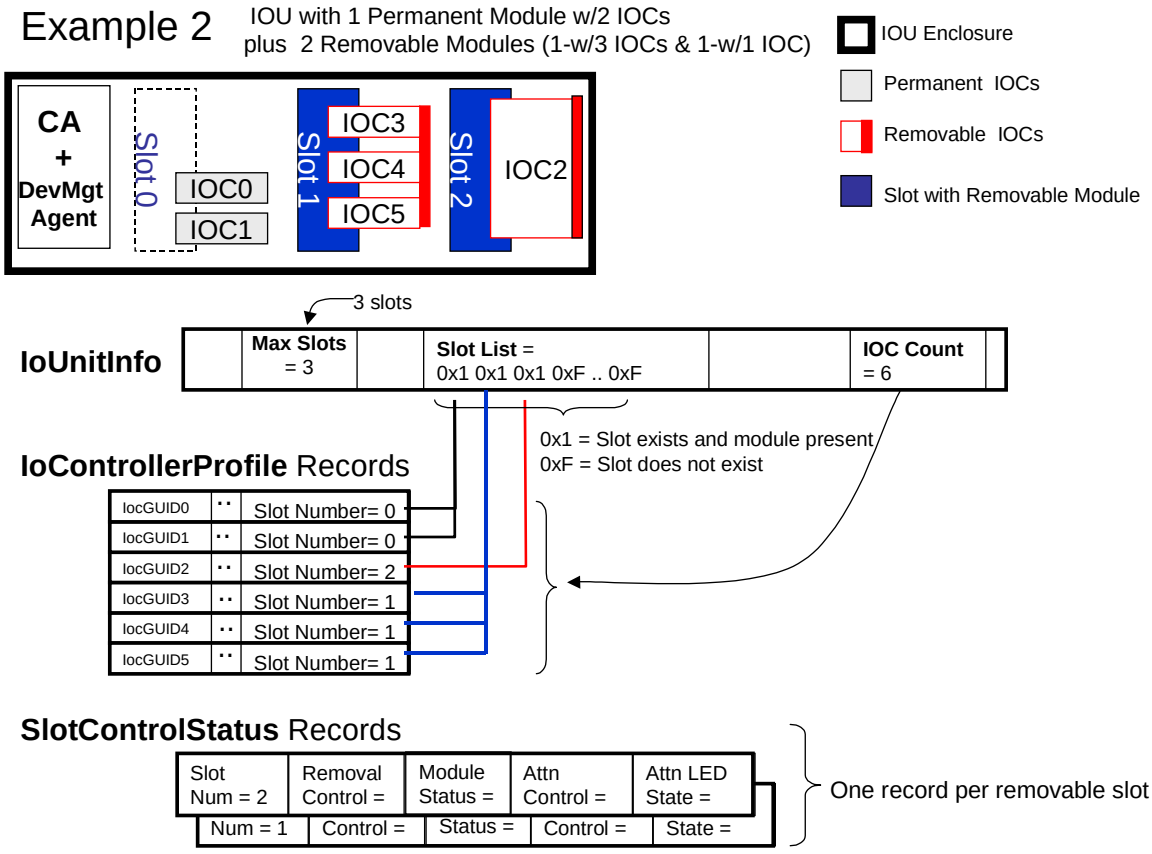


Figure 314 Example of IOU with Removable Modules

A8.2.3.5 DEVICE DIAGNOSTICS

As illustrated in Figure 315, the diagnostic framework provides the means for a diagnostic application to request a diagnostic session. This request is sent to the DM [step 1]. The manager validates the request, and if the source is authorized to perform diagnostics, the manager informs client platforms that will be affected by the diagnostics [steps 2,3]. establishes a diagnostic session with the IOU [step 4], and then informs the diagnostic application that it may proceed [step 5].

The diagnostic application performs diagnostics directly with the IOU [step 6] and when it is done running diagnostics, it releases the session by informing the manager [step 7]. The manager cancels the IOU's diagnostic session [step 8] and then notifies the affected clients that the I/O resources are back on-line [step 9].

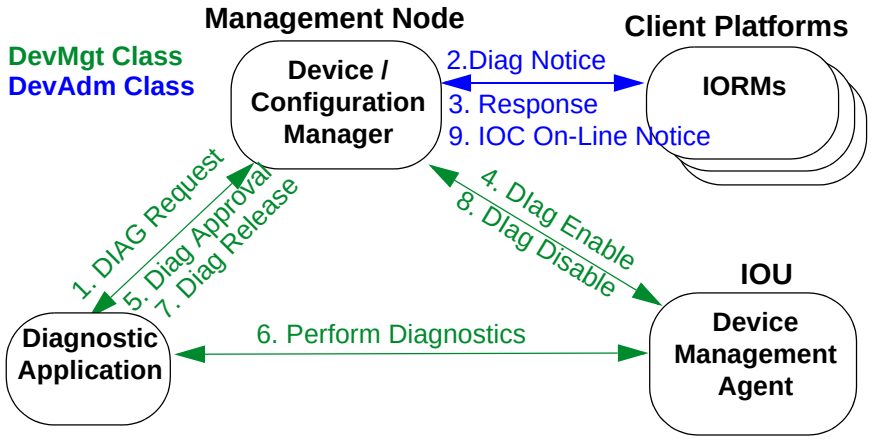


Figure 315 Diagnostic Usage Model

Steps 1,4,5,6,7, and 8 are performed under the DevMgt class and are defined in this annex. Steps 2,3, and 9 are performed under DevAdm class and are defined in Annex A7: Configuration Management.

In the initial session request [step 1] the diagnostic application indicates the severity and scope of the diagnostics. The manager uses the scope to determine which I/O clients to notify. The request also establishes a lease period for the session and the means to renew the lease if more time is needed.

The session setup [step 4] establishes the parameters for the diagnostic testing including severity and scope, and establishes the Token that the diagnostic application uses in diagnostic requests [step 6] to validate that it is authorized to perform diagnostics. The IOU rejects any diagnostic tests that violate the parameters of the diagnostic session.

Because it may take time for the IOU to prepare for diagnostic testing, the IOU replies immediately to the Diagnostic Session request and then sends a trap when it is prepare to perform diagnostics. That trap is passed on to the diagnostic application to indicate that the application may invoke diagnostic tests. The application invokes tests to be executed once or run continuously.

A8.2.4 LEVELS OF ACCESS

There are several levels of access to Device Management attributes.

Some attributes are available to anyone that can access the IOU.

All attributes are available to the DM and certain attributes are only accessible by the DM. The DevMgt Agent uses the Access_Key and KeyType

components in the MAD header to validate if a MAD comes from a valid DM (i.e., passes DevMgt key check), a supervisor, or a client.

Some attributes are available to the DM and specific supervisors. For example, the PlatformPoolRecord can only be accessed by the DM and the supervisor that the DM specifies in the PlatformPoolRecord.

Some attributes are only available to the DM, specific platform supervisors, and configured clients. For example, in addition to the DM:

- A ClientPoolRecord can only be read by the specific client and the client's supervisor, which may also write the record.
- ServiceRecords are restricted to clients and supervisors that have a corresponding ClientPoolRecord or PlatformPoolRecord specifying the ServiceObject (see below).
- ProtocolRecords and IOControllerProfile attributes are limited to clients and supervisors that have a corresponding ClientPoolRecord or PlatformPoolRecord specifying a ServiceObject of that IOC (see below).

The DM, using the PlatformPoolRecord attribute, programs the IOU with Supervisor_Keys, and using the ClientPoolRecord attribute, programs the IOU with Client_Keys.

A supervisor (IORM) must get its key from the DM. With the Supervisor_Key, the IORM can learn and the Client_Keys for its platform. Thus, clients get their keys from their IORM.

For a KeyType of Supervisor or Client, the DevMgt Agent matches the Access_Key in the MAD header to a corresponding PlatformPoolRecord or ClientPoolRecord respectively, which specifies the access rights.

- When a client requests attributes about service objects, the DevMgt Agent only returns records for service objects specified in the ClientPoolRecord with that Client_Key.
- When a supervisor requests attributes about service objects, the DevMgt Agent only returns records for service objects specified in the PlatformPoolRecord with that Supervisor_Key.

Some attributes are only available to the DM and specific diagnostic applications. For example:

- A DiagSession attribute can only be read by the DM and a diagnostic application with a valid DiagToken.
- TestDeviceOnce and TestDeviceLoop can only be set by the DM or a diagnostic application with a valid DiagToken.

The DM, using the DiagSession attribute, programs the IOU with DiagTokens. A diagnostic application must get its DiagToken from the DM. With the DiagToken, the diagnostic application is able to invoke diagnostics, but only on the objects the DM specifies in the DiagSession attribute.

For a KeyType of DiagToken, the DevMgt Agent matches the Access_Key in the MAD header to a corresponding DiagSession record, which specifies the access rights.

A8.3 DEVICE MGT MAD SPECIFICATION

A8.3.1 MAD FORMAT

CA8-1: The datagrams for the DevMgt class **shall** conform to the MAD format and use as specified in 13.4 Management Datagrams on page 717 and further specified in Figure 316 Device Management MAD Format on page 1530 and Table 447 Device Management MAD Fields on page 1531 below.

.

Figure 316 Device Management MAD Format

Offset	Byte 0	Byte 1	Byte 2	Byte 3
0	Common MAD Header			
...				
20				
24	RMPP Header			
...				
32				
36	Access_Key			
40				
44	KeyType	reserved		
48	Reserved			
52				
56				
60	Change_ID		ComponentMask	
64	DevMgt Data			
...				
252				

Table 447 Device Management MAD Fields

Field[a]	Length	Description
Common MAD Header	24 bytes	Common MAD Header as described in 13.4.2 Management Datagram Format on page 718. MgtClass=0x06, ClassVersion=2
RMPP Header	12 bytes	RMPP header as described in 13.4.2 Management Datagram Format on page 718
Access_Key	8 bytes	Manager key, supervisor key, client key, or diagnostic token - a value that the DevMgt Agent uses to validate the source of certain management operations as described in A8.3.3.12 KeyInfo on page 1576. and A8.3.1.4.3 Client_Key on page 1537. A node uses the default value of zero unless it has been configured otherwise.
KeyType	1 byte	A value that specifies the nature of the sender and indicates they type of key provided in the Access_Key component. • 0x00 = Client or DevMgt Agent • 0x01 = Supervisor • 0x02 = DM • 0x03 = Diagnostic Token
reserved	15 bytes	reserved
Change_ID	2 bytes	Count that indicates when one or more read-only components in an attributes has changed. It is incremented, with rollover, by any change to a Read-Only component of IOUnitInfo, IOControllerProfile, ProtocolRecord, ServiceRecord, ProductInfo, or IouResourceInfo attributes. Multiple changes to the same attribute should not result in multiple increments, while changes to multiple attributes should result in an increment for each attribute record modified, added, or deleted.
ComponentMask	2 bytes	Indicates which attribute components in the DevMgtGet() are to be used for the query. Refer to A8.3.1.5 Component Mask on page 1538
DevMgt Data	192 bytes	192 bytes of DevMgt payload. The structure and content depends upon the Method, Attribute, and Attribute Modifier fields in the MAD header.

a. The term 'MAD header' refers to all fields except the DevMgt Data field

CA8-2: The datagrams for the DevMgt class **shall** conform to the Common MAD requirements as specified in 20.14 "Common Mad Requirements".

A8.3.1.1 CLASS VERSION

Class version 2 provides significant enhancements over class version 1. Class version is indicated in two places. One is the ClassVersion field in the MAD header and the other is the ClassVersion component in the ClassPortInfo attribute. The following rules apply to IOUs and DMs supporting Class Version 2 as the highest version supported.

Table 448 Class Version

Received MADHeader: ClassVersion	Method/Attribute	Action[a]	Response MADHeader: ClassVersion	ClassPortInfo: ClassVersion
0 or >2	Get(ClassPortInfo)	respond	2	2
0 or >2	Set(ClassPortInfo)	reject	2	n/a
0 or >2	other than Get/Set ClassPortInfo	reject	2	n/a
2	Get(ClassPortInfo) Set(ClassPortInfo)	respond	2	2
2	other than Get/Set ClassPortInfo	respond	2	n/a
1	Get(ClassPortInfo)	respond	1	2
1	Set(ClassPortInfo) - if class version 1 supported	Conditional	1	2
1	other than Get/Set ClassPortInfo - if class version 1 supported	Conditional	1	n/a
1	all except Get(ClassPortInfo) - when class version 1 not supported	reject	1	n/a

a. Note that there are other reasons why a MAD is rejected independent of the Class Version. **Reject** means report Status(4:2)=001 “Either the base version, or the class version, or the combination of the two is not supported”. **Conditional** means that the KeyInfo:BackwardCompatibilityLevel component determines if the IOU responds or rejects the MAD (see Table 449 DevMgt Backward Compatibility on page 1534) and DM policy determines how the DM responds.

CA8-3: A DevMgt Agent and DM shall respond to MADHeader:ClassVersion values as per Table 448: Class Version

A8.3.1.1.1 BACKWARD COMPATIBILITY

Class version 1 does not have the facilities for enforcement of device assignment and access control that is provided in class version 2. Depending on the environment and whether the IOU is being shared by multiple hosts, it may or may not be advantageous for an IOU to support class version 1. A primary concern is to prevent class version 1 from being a means for hosts to circumvent the protection provided by class version 2 while providing the ability for the IOU to function when hosts only support class version 1.

When all hosts support class version 2 or higher, there might be no need for an IOU to support class version 1. Thus, IOUs are not required to support multiple Class Versions, but for those that do, this annex defines the

means to identify if the IOU does support ClassVersion 1 and a means for the DM to configure how the IOU responds to ClassVersion 1 MADs.

A8.3.1.1.2 BACKWARD COMPATIBILITY LEVEL

There is an 'IsBackwardCompatibilitySupported' bit in the version 2 ClassPortInfo Capability component that indicates if the DevMgt Agent supports class version 1.

There are three levels of backward compatibility for an IOU that supports class version 1 & 2 and the DM selects the level via the KeyInfo attribute. The levels are:

- **No_v1_Access:** The DevMgt Agent rejects all ClassVersion 1 MADs as 'version not supported'. This is the only possibility for IOUs that do not support ClassVersion 1.
- **Full_v1_Access**: The DevMgt Agent processes ClassVersion 1 MADs without any filtering of information based on ClassVersion 2 configuration. This is the default for IOUs that support ClassVersion 1, so they will function in a ClassVersion 1 environment that does not have a DM.
- **Programmed_v1_Access**: The DM uses a ClassVersion 2 Client Pool Table record to limit what information can be obtained via ClassVersion 1 MADs. This level allows the DM to program the IOU as to which service objects are accessible to hosts that use ClassVersion 1.

For '*Programmed_v1_Access*', there is a V1ClientKey component in the KeyInfo attribute that the DM sets to indicate which Client Pool Table defines the service objects accessible via class version 1 MADs. When the DevMgt Agent receives a ClassVersion 1 request, it treats the request as if it included a Client Access Key with this value. Thus, the client pool table record whose Client_Key matches V1ClientKey (a.k.a. the Ver1ClientPoolRecord) determines which service objects that any host using ClassVersion 1 can access and it also limits the resources (QPs, SLs, etc.) that all ClassVersion 1 hosts (together as a group) may consume.

Table 449: DevMgt Backward Compatibility specifies how the DevMgt Agent responds to ClassVersion 1 DevMgt MADs per the different BackwardCompatibilityLevel settings.

Table 449 DevMgt Backward Compatibility

ClassVersion 1 Attribute Name[a]	I/O Unit's Backward Compatibility Level Setting[b]		
	No_v1_Access	Full_v1_Access	Programmed _v1_Access
ClassPortInfo	Get() only - reject set() as 'version not supported'	Get/Set()	Get() only - reject set() as 'version not supported'
Notice	no ClassVersion 1 traps	v1 traps	no ClassVersion 1 traps
IOUnitInfo	*Reject*	Get() - controller list includes all IOCs	Get() - controller list only includes IOCs listed in Ver1ClientPoolRecord
IOControllerProfile	*Reject*	Get() - any IOCs	Get() - only for IOCs listed in Ver1ClientPoolRecord
ServiceEntries	*Reject*	Get() - any service object	Get() - only includes Service Objects listed in Ver1ClientPoolRecord
DiagnosticTimeout	*Reject*	Get()	*Reject*
PrepareToTest	*Reject*	Get/Set()	*Reject*
TestDeviceOnce	*Reject*	Set()	*Reject*
TestDeviceLoop	*Reject*	Set()	*Reject*
DiagCode	*Reject*	Get()	Get()
VendorSpecific	*Reject*	Get()	*Reject*

a. This table describes how a DevMgt Agent that supports class version 2 responds to class version 1 requests depending on how the DM had programmed the agent's BackwardCompatibilityLevel. Since the attribute in the response must match the version, any response must use version 1 attribute formats.
b. Reject means respond with Status bits 4:2 =1; "Bad version. Either the base version, or the class version, or the combination of the two is not supported".

A8.3.1.1.3 BACKWARD COMPATIBILITY REQUIREMENTS

CA8-4: If the DevMgt Agents sets the IsBackwardCompatibilitySupported bit in ClassPortInfo, then the agent shall support all Backward Compatibility Levels (No_v1_Access, Full_v1_Access, and Programmed _v1_Access).

CA8-5: If the DevMgt Agents does not set the IsBackwardCompatibilitySupported bit in ClassPortInfo, then it shall only support the 'No_v1_Access' Backward Compatibility Level. Thus, it shall ignore the BackwardCompatibilityLevel component in a DevMgtSet(KeyInfo) and always return a value of 'No_v1_Access' for the KeyInfo:BackwardCompatibilityLevel component.

CA8-6: If KeyInfo:BackwardCompatibilityLevel is 'No_v1_Access' and the DevMgt Agent receives a ClassVersion 1 DevMgt MAD, it shall reject the MAD with Status bits 4:2 =001b; "Bad version. Either the base version, or the class version, or the combination of the two is not supported".

oA8-1: If BackwardCompatibilityLevel is set to 'Full_v1_Access' and the DevMgt Agent receives a ClassVersion 1 DevMgt MAD, the agent shall respond as per Table 449: DevMgt Backward Compatibility using attribute formats defined in Section 16.3.

[Editorial Note:When the IOU is configured for 'Full_v1_Access'*, pool tables do not provide enforceable assignment of I/O resources to clients because any client could use version 1 DevMgt MADs to bypass the enforcement.]*

oA8-2: If BackwardCompatibilityLevel is set to 'Programmed _v1_Access' and the DevMgt Agent receives a ClassVersion 1 DevMgt MAD, the agent shall respond as per Table 449: DevMgt Backward Compatibility, using attribute formats defined in Section 16.3, but shall only provide information permitted by the ClientPoolRecord whose Client_Key matches the KeyInfo:Ver1ClientKey.

A8.3.1.2 STATUS FIELD

The Status field is described in 13.4.7 Status Field on page 731. Class-specific bits are defined in Table 450: Device Management Status Field.

Table 450 Device Management Status Field

Bits [8:15]	Name	Meaning
0x00	Operational	no additional status
0x01	IocErrorDetected	IOC not responding or in a failed state
0x02	NoMatchingRecord	The query did not result is a match
0x04	TableFull	Cannot create record because table capacity has been reached
0x80	GeneralFailure	IOU General Failure
0x10	InvalidDiagObject	Object not part of DiagSession:DiagScope
0x11	DiagSeverityLimit	Test exceeds DiagSession:DiagSeverity level
0x12	InvalidDiagTest	Test is undefined or not supported (TestType +TestTargetType + VendorSpecific)
0x40	NotMasterDM	This is not the master DM
0x41	IncompatibleVersion	The requested operation is not valid due to the class version of the target IOU.

Table 450 Device Management Status Field (Continued)

Bits [8:15]	Name	Meaning
0x42	PolicyReject	Rejected because of Device Manager Policy

A8.3.1.3 RMPP HEADER

DevMgt uses the Reliable Multi-Packet Protocol as described in Section 13.6 “Reliable Multi-Packet Transaction Protocol”. This protocol allows the initiator to make a single query which returns more information than can be transferred in a single packet. For example, an IORM can make a single request for IOControllerProfile attributes and have the DevMgt agent return all of the IOControllerProfile records. The response includes multiple DevMgtGetResp(IOControllerProfile) packets if necessary. Only the method/attribute combinations listed as ‘RMPP’ in Table 453, “DevMgt Agent Attribute / Method Map,” on page 1542 use the Reliable Multi-Packet Protocol and thus have multi-packet responses. DevMgtGet() requests are always single packets.

CA8-7: A DevMgt agent shall implement Reliable Multi-Packet Protocol as described in Section 13.6 “Reliable Multi-Packet Transaction Protocol.

CA8-8: A DevMgt agent shall not respond with more than one packet for method/attribute combinations not listed as ‘RMPP’ in Table 453, “DevMgt Agent Attribute / Method Map,” on page 1542.

CA8-9: A DevMgt agent shall reject a request that contains more than one packet except for a DevMgtSet() for an attribute listed as RMPP in the DevMgtSet() column of Table 453 on page 1542.

A8.3.1.4 ACCESS_KEY AND KEYTYPE

The Access_Key and KeyType components in the MAD:Header are used to validate the source of the MAD. The KeyType component indicates if the MAD is from a client, supervisor, DM, or diagnostic program and the Access_Key component provides the Client_Key, Supervisor_Key, Manager_Key, or DiagToken respectively.

The Access_Key validates the requestor’s level of access. The RAL and WAL columns of Table 453 on page 1542 specifies the access level permitted for each attribute. Some attributes can only be read or set by the DM, which can get and set all attributes. A supervisor is only allowed to get and set certain records of certain attributes, and a client can only get a subset of its supervisor records.

The DevMgt Agent always specifies a MADHeader:Access_Key value of zero in DevMgt MADs it sends.

CA8-10: The DevMgt agent shall set the MADHeader:Access_Key component to zero when generating a DevMgtTrap().

CA8-11: The DevMgt agent shall set the MADHeader:Access_Key component to zero when sending a DevMgtGetResp().

CA8-12: The DevMgt agent shall ignore the value of the MADHeader:Access_Key component in a DevMgtTrapRepress().

Note that the DevMgt agent validates a TrapRepress() by the TransactionID and Notice attribute content, which is only known by the recipient of the trap, which was set in ClassPortInfo by the DM.

A8.3.1.4.1 MANAGER_KEY

The IOU's Manager_Key is set by the DM to prevent others from modifying the configuration of the IOU (i.e., the IOU only accepts a DevMgtSet() from a manager with the correct Manager_Key, a supervisor with a correct Supervisor_Key, or a diagnostic application with the correct Diag Token).

A DM sets an IOU's Manager_Key and other manager key related parameters via the DevMgtSet(KeyInfo). See A8.3.3.12 KeyInfo on page 1576 for details on how the DevMgt agent uses the Manager_Key.

- The DevMgt agent only accepts DevMgtSet()s that contain a valid MADHeader:Access_Key (i.e., KeyType = Manager and Access_Key matches the key set via the KeyInfo attribute, KeyType = Supervisor and Access_Key matches a Supervisor_Key set by the DM, or KeyType = DiagToken and Access_Key matches a currently active DiagToken set by the DM).
- The DevMgt Agent also uses the MADHeader:Access_Key to validate if a DevMgtGet() came from a valid DM.

A8.3.1.4.2 SUPERVISOR_KEY

The Supervisor_Key identifies the privileged user on a particular client platform (i.e., the platform's IORM) that may configure that platform's client resource pools. A DM sets Supervisor_Keys in platform pool table records and client pool table records. The platform pool table record lists the service objects that the platform is permitted to use. The DevMgt Agent limits a supervisor's access to reading its own platform pool table record, to reading and writing its own client pool table records, and reading information about IOCs and service objects that the platform is permitted to use.

A8.3.1.4.3 CLIENT_KEY

Each client pool record has a Client_Key which identifies the client that is permitted to use the listed service objects and specified resources

(Number of QPs, client priority, service levels, etc.). The DevMgt Agent limits a client's access to getting its own client pool table record and information about IOCs and service objects that the client is permitted to use.

A8.3.1.4.4 DIAGTOKEN

The DM programs the IOU with a Diagnostic Token (DiagToken) specifying the scope and severity level of diagnostics that may be performed using that token. The manager provides the diagnostic program with that token. The diagnostic program specifies the DiagToken in diagnostic requests that it sends to the IOU to validate that the program is authorized to perform the diagnostics. When the DevMgt Agent receives a Set() for a diagnostic attribute, the agent validates that the DiagToken in the MAD matches a valid DiagToken and that the request is within the scope specified by that token.

CA8-13: If the DevMgt agent receives a DevMgt MAD with a MADHeader:KeyType = Diagnostic Token the MAD specifies an attribute that does not have a value of 'D' or 'A' in the respective RAL or WAL column of Table 7 on page 43, then the DevMgt agent shall reject the MAD.

A8.3.1.5 COMPONENT MASK

The ComponentMask in the MAD header is used in queries for attributes supporting RMPP (see Table 453 DevMgt Agent Attribute / Method Map on page 1542). The ComponentMask allows the initiator to indicate which components in the query (i.e., the DevMgtGet request) that the DevMgt agent uses in determining which records to return. Each bit corresponds to an attribute component as specified in each attribute's definition in A8.3.3: Attributes. If that bit is set to zero, the DevMgt agent ignores that component when selecting which attributes to return. When the bit is one, the DevMgt agent only returns records that have that component matching the value supplied in the request. For example, a DevMgtGet(ProtocolRecord) with component mask bits for Category, OrgID, and Protocol set to one will return all protocol records matching the Category, OrgID, and Protocol values supplied in the attribute in the DevMgtGet(ProtocolRecord) query.

CA8-14: The DevMgt agent shall ignore ComponentMask in MADs not indicated as 'RMPP' in Table 453, "DevMgt Agent Attribute / Method Map," on page 1542

CA8-15: The DevMgt agent shall ignore ComponentMask bits that are not defined in the attribute's definition in Section 8.3.3, "Attributes," on page 1540

CA8-16: For method/attribute combinations marked 'RMPP' in Table 453, "DevMgt Agent Attribute / Method Map," on page 1542, the DevMgt agent

shall only return attribute records matching the components indicated by the ComponentMask in the request.

CA8-17: The DevMgt agent shall set the ComponentMask in the response to the same value as in the request, except that any ComponentMask bits not supported (i.e., not defined for the attribute in this version) shall be set to zero. This requires that the ComponentMask in responses to non RMPP method/attributes combinations is always returned as zero.

A8.3.2 METHODS

Each MAD specifies a Method and an Attribute type. Common methods are defined in Chapter 13. Table 451 Device Management Methods on page 1539 specifies the common methods used by Device Management.

Device Management attributes are listed in Table 452 Device Management Attributes on page 1540. Table 453 DevMgt Agent Attribute / Method Map on page 1542 specifies which attributes are valid with which methods for a DevMgt agent. Table 454 DM Attribute / Method Map on page 1543 specifies which attributes are valid with which methods for a DM. The format of each attribute is defined in the subsections of A8.3.3 Attributes on page 1540.

The DevMgt Agent initially receives MADs through the GSI, which is an unreliable datagram service. The actual access QP and DLID may be redirected by the GSI.

Table 451 Device Management Methods

Method Type	Value	Description
DevMgtGet()	0x01	Request (read) DevMgt class attributes such as IOU information, IOC profiles, or service objects be returned. The response is a DevMgtGetResp().
DevMgtSet()	0x02	Request (write) an attribute to be set. The response is a DevMgtGetResp().
DevMgtGetResp()	0x81	Response to a DevMgtGet() or DevMgtSet() request.
DevMgtTrap()	0x05	Unsolicited datagram sent to the DM. Contains the Notice attribute as defined in A8.3.3.2 Notice on page 1545 to identify the trap.
DevMgtTrapRepress()	0x07	Cancel repetition of notification.
DevMgtReport()	0x06	Forward an event to an entity that previously subscribed for it
DevMgtReportResp()	0x86	Reply to a DevMgtReport()

A8.3.3 ATTRIBUTES

This section specifies the format of the attributes used for managing the IOU.

Device Management attributes are listed in Table 452 Device Management Attributes on page 1540.

Table 453 DevMgt Agent Attribute / Method Map on page 1542 specifies which attributes are valid with which methods for a DevMgt agent, identifies which are required to be supported, and specifies which method/attribute combination can be multi-packet. It also indicates if the attribute can only be accessed by the DM, by configured clients, or by anyone.

Table 452 Device Management Attributes

Attribute Name	Attribute ID[a]	Scope[b]	Description
ClassPortInfo page 1544	0x0001	Port	Provides information about the DevMgt Agent and allows the DM to program where the DevMgt Agent sends traps. See A8.3.3.1 ClassPortInfo on page 1544
Notice page 1545	0x0002	IOU	Provides Trap details. See A8.3.3.2 Notice on page 1545
InformInfo page 1557	0x0003	DM	Attribute for interested parties to subscribe with the DM for DevMgt Traps.
IOUnitInfo page 1559	0x0010	IOU	Information about the IOU and lists status of subassembly slots. Each IOU may support up to 255 subassemblies. Each subassembly can have an unlimited number of IOCs and each IOC can have 2^{16} service objects.
IOControllerProfile page 1561	0x0011	IOU	IOC Profile Information. Note: Attribute Modifier no longer identifies the IOC. The IOU maintains a table of IOControllerProfile records. The ComponentMask may be used to request a particular IOControllerProfile record or a particular set of IOControllerProfile records. If all IOControllerProfile records are needed, the requester sets the component mask to zero.
ServiceEntries	0x0012	n/a	The ServiceEntries attribute has been replaced by the ServiceRecord attribute and is no longer used
DA Info page 1558	0x0013	DM	Provides the address of the Device Administrator
ServiceRecord page 1564	0x0014	IOU	Specifies a Service ID for communicating with a service object. The IOU maintains a table of ServiceRecords for the IOU and all its IOCs. The ComponentMask may be used to request a particular ServiceRecord or a particular set of ServiceRecords, or set to zero to request all records.
ProtocolRecord page 1568	0x0015	IOU	Specifies a protocol (management protocol or I/O protocol) supported by the IOU. The ComponentMask may be used to request a particular protocol record or a particular set of protocol records, or set to zero to request all records.

Table 452 Device Management Attributes (Continued)

Attribute Name	Attribute ID[a]	Scope[b]	Description
SlotControlStatus page 1570	0x0016	IOU	Get()s and Set()s information pertaining to the removal state of a subassembly. The ComponentMask may be used to request a particular SlotControlStatus or a particular set of SlotControlStatus records, or set to zero to request SlotControlStatus on all slots.
KeyInfo page 1576	0x0017	IOU	Information pertaining to the Manager_Key used to check received DevMgt class MADs
Reset page 1573	0x0018	IOU	Used to reset the IOU or an IOC
ProductInfo page 1574	0x0019	IOU	Provides product specific information about the IOU or an IOC.
reserved	0x001A-0x001F	n/a	reserved
DiagnosticTimeout page 1602	0x0020	IOU	Provides the maximum time for completion of diagnostic test. Tests not completing within this period may indicate device failure.
PrepareToTest	0x0021	n/a	The 'PrepareToTest' attribute has been replaced by the DiagSession attribute and is no longer used
TestDeviceOnce page 1603	0x0022	IOU	A DevMgtSet() of this attribute instructs the IOU to initiate the diagnostic test specified in the attribute and run it once.
TestDeviceLoop page 1604	0x0023	IOU	A DevMgtSet() of this attribute instructs the IOU to initiate the diagnostic test specified in the attribute and run it continuously in a loop.
DiagCode page 1605	0x0024	IOU	Vendor-specific diagnostic information for the device specified in the attribute.
DiagSession page 1598	0x0025	IOU / DM	A DevMgtSet() of this attribute sent to the DM requests a diagnostic session. When sent to the IOU by the DM, instructs the IOU to prepare the object(s) specified by the attribute for diagnostic testing.
reserved	0x0026 - 0x003F	n/a	reserved
IouResourceInfo page 1582	0x0040	IOU	Specifies resource capabilities of the IOU.
PlatformPoolRecord page 1585	0x0041	IOU	Permits the DM to specify which service objects and resources a platform may use.

Table 452 Device Management Attributes (Continued)

Attribute Name	Attribute ID[a]	Scope[b]	Description
ClientPoolRecord page 1591	0x0042	IOU	Permits the platforms' supervisor to specify which of service objects and resources a client may use.
KeyChange page 1597	0x0043	IOU	Changes a Client_Key or Supervisor_Key'
reserved	0x0044-0xFEFF	n/a	reserved
Vendor-specific	0xFF00-0xFFFF	vendor defined	Vendor-unique attribute values may be defined to deliver specific test instructions.

a. Attribute Modifier field is not used and therefore is set to zero. Vendor-Specific attributes may use Attribute Modifiers.
b. Scope identifies whether each port has its own set of that attribute (i.e., Port scope) or the IOU maintains a single set of that attribute accessible via any of its ports (i.e., IOU scope).
For IOU scope, a DevMgtSet() on any port changes the attribute content for all ports such that a DevMgtGet() on any port returns the same information.
For Port scope, a DevMgtSet() on one port does not change the attribute content for other ports.
For DM scope, if the DM uses multiple QPs to receive DevMgtSet()s, the DM treats the requests as if they came to the same QP regardless of whether they come through the same port or not.

Table 453 DevMgt Agent Attribute / Method Map[a]

Attribute Name	RAL[b]	WAL[c]	DevMgtGet()[d]	DevMgtSet()[d]	DevMgtTrap()	TrapRepress()
ClassPortInfo	A	M	***yes***	***yes***		
Notice	M	A	yes	yes	***yes***	***yes***
IOUnitInfo	A	-	***yes***			
IOControllerProfile	MSCD	-	***RMPP***			
ServiceRecord	MSC	-	***RMPP***			
ProtocolRecord	MSC	-	***RMPP***			
SlotControlStatus	A	M	***RMPP***	yes		
ProductInfo	A	-	RMPP			
KeyInfo	A	M	***yes***	***yes***		
Reset	A	M	***yes***	***yes***		
IouResourceInfo	M	M	***RMPP***	***yes***		
PlatformPoolRecord	MS	M	***RMPP***	***yes***		
ClientPoolRecord	MSC	MS	***RMPP***	***yes***		
KeyChange		M		***yes***		

Table 453 DevMgt Agent Attribute / Method Map[a] (Continued)

Attribute Name	RAL[b]	WAL[c]	DevMgtGet()[d]	DevMgtSet()[d]	DevMgtTrap()	TrapRepress()
DiagnosticTimeout	A	-	***yes***			
DiagSession	MD	M	***RMPP***	***yes***		
TestDeviceOnce	-	MD		***yes***		
TestDeviceLoop	-	MD		***yes***		
DiagCode	A	-	***yes***			

a. Combinations in <***Bold Italic***> are required to be supported by all DevMgt agents.
b. Read Access Level - RAL identifies who the DevMgt Agent allows to read the attribute. A=all, M=DM, S=configured supervisors, C=configured clients (see A8.4.2 Filtering Information on page 1608), D=Diagnostic program.
c. Write Access Level - The WAL identifies who the DevMgt Agent allows to write the attribute.
d. RMPP designates that the method/attribute uses Reliable Multiple Packet Protocol. Note that the DevMgtGet() is a single packet that can result in a multiple packet response.

The attributes are requested and returned through the GSI, which is an unreliable datagram service. The actual access QP and DLID may be re-directed by the GSI.

In addition to the MADs that the DM sends to DevMgt Agents, the DM is required to support additional Method-Attribute combinations. Table 454 DM Attribute / Method Map on page 1543 specifies which attributes are valid with which methods for a DM and identifies which are required to be supported.

Table 454 DM Attribute / Method Map[a]

Attribute Name	DevMgtGet()	DevMgtSet()	DevMgtReport()
ClassPortInfo	***yes***		
InformInfo		***yes***	
Notice			***yes***
DA Info	***yes***		
DiagSession	***yes***	***yes***	
ProductInfo	yes		

a. Combinations in <***Bold Italic***> are required to be supported by a DM.

The DM registers with the SA using SubAdmSet(ServiceRecord). The QP the manager uses to receive these manager MADs can be learned via a CM:SIDR_REQ using the ServiceID specified in the SA:ServiceRecord. The QP is an unreliable datagram service.

A8.3.3.1 CLASSPORTINFO

The ClassPortInfo attribute is described in 13.4.8.1 ClassPortInfo on page 734. The CapabilityMask component provides for class-specific bits which are defined in Table 455: Device Management Agent ClassPortInfo:CapabilityMask and Table 456: Device Manager ClassPortInfo:CapabilityMask.

Table 455 Device Management Agent ClassPortInfo:CapabilityMask

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.8.1 ClassPortInfo on page 734.
8	reserved	reserved
9	IsManager_KeyNonVolatile	Indicates if DevMgt Agent preserves its Manager_Key across power cycles. • 1b = Manager_Key saved in non-volatile storage • 0b = Manager_Key not saved in non-volatile storage
10	NonVolatileAttributes	Indicates if DevMgt Agent preserves R/W components of all attributes across power cycles. • 1b = Attributes saved in non-volatile storage • 0b = Attributes not saved in non-volatile storage
11	GracefulHotPlug	Indicates if DevMgt Agent supports graceful hot plug • 1b = Graceful hot plug supported • 0b = Graceful hot plug not supported
12	IsBackwardCompatibilitySupported	When set indicates that the DevMgt agent supports multiple Class Versions (i.e., v1 and v2). Requirements for setting this bit are specified in A8.3.1.1.3: Backward Compatibility Requirements.
13-15	-	reserved

Device Managers have different capabilities than Device Management Agents. Thus, Table 456 specifies CapabilityMask bits for a DM.

Table 456 Device Manager ClassPortInfo:CapabilityMask

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.8.1 ClassPortInfo on page 734. Note that bits 0 and 1 are not meaningful for a class manager and thus are set to zero and ignored.
8-13	reserved	reserved
14	GracefulFailover	This bit indicates if the DM shares subscription and DIag Session information with standby mangers. A value of 1 indicates that the DA retains subscriptions and DIag Session information across fail-overs. Requirements for setting this bit are specified in Annex A7: Configuration Management in sections A7.5.7.3 Subscription Integrity on page 1461 and A7.5.9.1 Diagnostic Framework on page 1468.
15	IsContextPersistent	This bit indicates if the DM persistently stores subscription and DIag Session information such that subscriptions are retained across reset, restarts, and power cycles. Requirements for setting this bit are specified in Annex A7: Configuration Management in sections A7.5.7.3 Subscription Integrity on page 1461 and A7.5.9.1 Diagnostic Framework on page 1468.

A8.3.3.2 NOTICE

The Notice attribute is described in 13.4.8.2 Notice on page 737. It is used for generic traps as specified in Table 457: Notices for Device Management Traps.

Trap datagrams are described in 13.4.9 Traps on page 741.

CA8-18: When the DevMgt agent generates a trap, then that trap shall be issued from all ports that have a non-zero ClassPortInfo:TrapLID.

CA8-19: Upon receipt of a valid TrapRepress() MAD and independent of the MADHeader:Access_Key, the DevMgt agent shall cease sending the trap which matches the trap identified by the TrapRepress() MAD. A trap being repeatedly sent matches a trap identified in a TrapRepress() MAD when both MADHeader:TransactionID in the trap MAD matches MADHeader:TransactionID in the TrapRepress MAD and the Notice attribute in the trap MAD matches the Notice attribute in the TrapRepress().

A DevMgtTrap() conveys a Notice attribute and the Notice attribute contains a DataDetails component, which provides trap specific information. Thus, Device Management defines the DataDetails for each type of trap.

In each DataDetails definition there is a LostTrap bit. This bit is used to signal when the IOU had to stop sending a trap before receiving a DevMgtTrapRepress(). The premise is that the DevMgt Agent has a trap queue of limited depth (one or more outstanding traps). When a new trap is generated, it is placed at the tail of the trap queue and the trap at the head of the queue is repeated until the DevMgt Agent receives a matching DevMgtTrapRepress(). The matching trap is then discarded and then next trap in the queue is sent immediately and repeated until a matching DevMgtTrapRepress is received.

If the trap queue is full and a new trap is generated, the DevMgt Agent discards the trap at the head of the trap queue, adds the new trap to the tail of the trap queue, sets the LostTrap bit in trap at the head of the queue and immediately starts sending that trap.

There could be other reasons why a trap is discarded before it is acknowledged. It is the DevMgt Agent's responsibility to set the LostTrap bit in the first trap sent after the discarded trap would have been sent. If the discarded trap is the only trap in the queue, then the agent should fabricate a Status Report trap with StatusType = 0x01 (Trap Queue Flushed) and the LostTrap bit set to indicate the lost trap.

There might be other events that could cause the agent to prematurely discard one or more traps. When there is at least one trap to be reported and a condition occurs in which the DevMgt Agent discards all of the traps, then the DevMgt Agent generates a StatusReport trap with a StatusType of 0x01 and the LostTrap bit set. This will inform the Device Manager and subscribed clients that one or more traps may have been lost.

Unless otherwise specified, Notice attribute components for Notices specified in Table 457 shall be set as follows:

1) IsGeneric = 1 (Generic)
2) Type - as per Table 457
3) ProducerType = 1 (Channel Adapter)
4) TrapNumber - as per Table 457
5) IssuerLID - as per 13.4.8.2 Notice on page 737
6) NoticeToggle = 0 as per 13.4.8.2 Notice on page 737
7) Notice Count = 0 as per 13.4.8.2 Notice on page 737
8) DataDetails - as per Table 457
9) IssuerGID=0 as per 13.4.8.2 Notice on page 737

Table 457 Notices for Device Management Traps

Name	Type	Number	DataDetails (padded to 432 bits)
MgrKeyViolation	Security	0x0000	Reports an Access_Key violation as per Table 458 Notice 0x0000 DataDetails [MgrKey Violation] on page 1548
SupvKeyViolation	Security	0x0001	Reports an Access_Key violation as per Table 459 Notice 0x0001 DataDetails [SupvKey Violation] on page 1549
Client Violation	Security	0x0002	Reports an Access_Key violation as per Table 460 Notice 0x0002 DataDetails [Client Violation] on page 1550
DiagToken Violation	Security	0x0003	Reports the use of an invalid DiagToken as per Table 461 Notice 0x0003 DataDetails [DiagToken Violation] on page 1551
Heartbeat	Informational	0x0007	Indicates that the DM is still active as per Table 462 Notice 0x0007 DataDetails [Heartbeat] on page 1552
StatusReport	Informational	0x0008	Status change as per Table 463 Notice 0x0008 DataDetails [Status-Report] on page 1552
IO Controller Change	Informational	0x0010	IOC with <IocGUID> has been <ACTION = added, deleted, or modified> as per Table 464 Notice 0x0010 DataDetails [IO Controller Change] on page 1553
Service Object Change	Informational	0x0011	ServiceRecord with for <IocGUID> with <ServiceObjectID> has been <ACTION = added, deleted, or modified> as per Table 465 Notice 0x0011 DataDetails [ServiceRecord Change] on page 1554
Slot Status Change	Informational	0x0018	Status of I/O module with <SlotNumber> has changed to <SlotStatus> as per Table 466 Notice 0x0018 DataDetails [Slot Status Change] on page 1555
IomRemoval	Informational	0x0020	Request to remove I/O module <SlotNumber> as per Table 467 Notice 0x0020 DataDetails [IomRemoval] on page 1555
ReadyToTest	Informational	0x0202	The ReadyToTest notice (514) has been replaced with the DiagSessionState notice and is no longer used
DiagSessionState	Informational	0x0801	Diagnostic session <DiagToken> readiness is <DiagSessionStatus>, where DiagSessionStatus is the same as would have been returned by a DevMgtGetResp(DiagSession) for that DiagToken as per Table 468 Notice 0x0801 DataDetails [DiagSessionState] on page 1556.
DiagSession Violation	Security	0x0802	Reports a diagnostic violation as per Table 469 Notice 0x0802 DataDetails [DiagSession Violation] on page 1556

CA8-20: A DevMgt agent shall use the layouts for the DataDetails component specified in A8.3.3.2: Notice for corresponding Traps. Fields **shall** be filled with the information corresponding to the description of a given trap.

A8.3.3.2.1 MGRKEY VIOLATION NOTICE

The IOU uses the MgrKey Violation notice to inform the DM that a it received a DevMgt MAD with an invalid Manager_Key.

Table 458 Notice 0x0000 DataDetails [MgrKey Violation]

Field	Offset (bits)	Length (bits)	Description
reserved	0	8	reserved
Method	8	8	Method used in MAD that caused the violation
AttributeID	16	16	AttributeID used in MAD that caused the violation
ViolatedPortGUID	32	64	Port GUID of the port receiving the violation
OffendingPortGid	96	128	Requestor port GID from the MAD that caused the violation
OffendingLIDADDR	224	16	SLID used in MAD that caused the violation
reserved	240	16	reserved
QP	256	24	Source Queue Pair from the MAD that caused the violation
reserved	280	8	reserved
Manager_Key	288	64	MADHeader:Access_Key from the MAD that caused the violation
Timestamp	352	48	Timestamp when violation was detected. The format of the timestamp is the number of milliseconds since 1/1/70 00:00am Zero indicates Timestamp not supported
reserved	400	31	reserved
LostTrap	431	1	This bit is set when the IOU had to discard a Trap without receiving a TrapRepress() for it. The bit is reset when the IOU receive a valid TrapRepress() with this bit set.

CA8-21: The DevMgt agent shall create the MgrKey Violation notice as specified in Table 458 and issue the Trap from every port when it detects a Manager_Key mismatch as per A8.3.3.12 KeyInfo on page 1576.

A8.3.3.2.2 SUPVKEY VIOLATION NOTICE

The IOU uses the SupvKey Violation notice to inform the DM that a it received a DevMgt MAD with an invalid Supervisor_Key (MADHeader:Key-

Type = Supervisor and no PlatformPoolRecord with a Supervisor_Key matching MADHeader:Access_Key).

Table 459 Notice 0x0001 DataDetails [SupvKey Violation]

Field	Offset (bits)	Length (bits)	Description
reserved	0	8	reserved
Method	8	8	Method used in MAD that caused the violation
AttributeID	16	16	AttributeID used in MAD that caused the violation
ViolatedPortGUID	32	64	Port GUID of the port receiving the violation
OffendingPortGid	96	128	Requestor port GID from the MAD that caused the violation
OffendingLIDADDR	224	16	SLID used in MAD that caused the violation
reserved	240	16	reserved
QP	256	24	Source Queue Pair from the MAD that caused the violation
reserved	280	8	reserved
Supervisor_Key	288	64	MADHeader:Access_Key from the MAD that caused the violation
Timestamp	352	48	Timestamp when violation was detected. The format of the timestamp is the number of milliseconds since 1/1/70 00:00am Zero indicates Timestamp not supported
reserved	400	31	reserved
LostTrap	431	1	This bit is set when the IOU had to discard a Trap without receiving a TrapRepress() for it. The bit is reset when the IOU receive a valid TrapRepress() with this bit set.

CA8-22: The DevMgt agent shall create the SupvKey Violation notice as specified in Table 459 and issue the trap to every port when it detects an invalid Supervisor_Key as per A8.4.2 Filtering Information on page 1608.

A8.3.3.2.3 CLIENT VIOLATION NOTICE

The IOU uses the Client Violation notice to inform the DM that a it received a DevMgt MAD or a CM MAD with an invalid Client_Key.

Table 460 Notice 0x0002 DataDetails [Client Violation]

Field	Offset (bits)	Length (bits)	Description
MADClass	0	8	MgmtClass value used in MAD that caused the violation - indicates if DevMgt or CM MAD.
Method	8	8	Method used in MAD that caused the violation
AttributeID	16	16	AttributeID used in MAD that caused the violation
ViolatedPortGUID	32	64	Port GUID of the port receiving the violation
OffendingPortGid	96	128	Requestor port GID from the MAD that caused the violation. Zero indicates that the MAD did not contain a GRH.
OffendingLIDADDR	224	16	SLID used in MAD that caused the violation
reserved	240	16	reserved
QP	256	24	Source Queue Pair from the MAD that caused the violation
reserved	280	8	reserved
Client_Key	288	64	MADHeader:Access_Key from the DevMgt MAD or the PrivateData:Client_Key from the CM MAD that caused the violation.
Timestamp	352	48	Timestamp when violation was detected. The format of the timestamp is the number of milliseconds since 1/1/70 00:00am Zero indicates Timestamp not supported
reserved	400	31	reserved
LostTrap	431	1	This bit is set when the IOU had to discard a Trap without receiving a TrapRepress() for it. The bit is reset when the IOU receive a valid TrapRepress() with this bit set.

CA8-23: The DevMgt agent shall create the Client Violation notice as specified in Table 460 and issue the Trap from every port when it detects a Client_Key mismatch as per A8.4.2 Filtering Information on page 1608.

CA8-24: The DevMgt agent shall create the Client Violation notice as specified in Table 458 and issue the Trap from every port when it detects a Client_Key mismatch as per A8.4.3 Restricting Access on page 1610.

A8.3.3.2.4 DIAGTOKEN VIOLATION NOTICE

The IOU uses the DiagToken Violation notice to inform the DM that a it received a DevMgt MAD with an invalid DiagToken.

Table 461 Notice 0x0003 DataDetails [DiagToken Violation]

Field	Offset (bits)	Length (bits)	Description
reserved	0	8	reserved
Method	8	8	Method used in MAD that caused the violation
AttributeID	16	16	AttributeID used in MAD that caused the violation
ViolatedPortGUID	32	64	Port GUID of the port receiving the violation
OffendingPortGid	96	128	Requestor port GID from the MAD that caused the violation. Zero indicates that the MAD did not contain a GRH.
OffendingLIDADDR	224	16	SLID used in MAD that caused the violation
reserved	240	16	reserved
QP	256	24	Source Queue Pair from the MAD that caused the violation
reserved	280	8	reserved
DiagToken	288	64	MADHeader:Access_Key from the DevMgt MAD that caused the violation.
Timestamp	352	48	Timestamp when violation was detected. The format of the timestamp is the number of milliseconds since 1/1/70 00:00am Zero indicates Timestamp not supported
reserved	400	31	reserved
LostTrap	431	1	This bit is set when the IOU had to discard a Trap without receiving a TrapRepress() for it. The bit is reset when the IOU receive a valid TrapRepress() with this bit set.

CA8-25: The DevMgt agent shall create the DiagToken Violation notice as specified in Table 461 and issue the Trap from every port when it receives a MAD with KeyType = DiagToken but the Access_Key does not match DiagToken of a current Diagnostic Session as per A8.3.3.17 DiagSession on page 1598.

A8.3.3.2.5 HEARTBEAT NOTICE

This notice is not generated by the IOU, it is generated by a DM in a DevMgtReport() to inform subscribed clients that the DM is still operational.

Table 462 Notice 0x0007 DataDetails [Heartbeat]

Field	Offset (bits)	Length (bits)	Description
TTNH	0	12	Time till next heartbeat - specifies the number of minutes before the DM will send another Heartbeat notice. If more that this time elapses, it is an indication that the DA has terminated the subscription.
reserved	12	4	
Fail-over	16	1	When this bit is set to one it indicates that a standby manager has taken-over. The client platform should query the SA to locate the new DM.
reserved	17		
reserved	431	1	The LostTrap bit is not used since this notice is not sent in a Trap().

See Configuration Management Annex A7.5.7.5 Heartbeat on page 1463 for requirements on generating the Heartbeat notice.

A8.3.3.2.6 STATUSREPORT NOTICE

This notice is generated by the IOU to inform the Device Manager and subscribed clients that there was a change in operational status. Currently this trap is only used to indicate that all outstanding traps have been discarded prior to receiving an acknowledge.

Table 463 Notice 0x0008 DataDetails [StatusReport]

Field	Offset (bits)	Length (bits)	Description
ReportType	0	8	Report Type • 0x01 = Trap Queue Flushed all other values reserved
reserved	0	423	reserved
LostTrap	431	1	This bit is set when the IOU had to discard a Trap without receiving a TrapRepress() for it. The bit is reset when the IOU receive a valid TrapRepress() with this bit set. It is always set when ReportType = ”Trap Queue Flushed”.

See Section 8.3.3.2 on page 1545 for requirements on generating the Status Report notice.

A8.3.3.2.7 IOC CHANGE NOTICE

The IOU uses the IO Controller Change Notice to inform the DM that an IOC has been installed, removed, or modified.

Table 464 Notice 0x0010 DataDetails [IO Controller Change]

Field	Offset (bits)	Length (bits)	Description
IocGUID	0	64	IOC GUID from IOControllerProfile attribute
reserved	64	64	reserved
ACTION	128	2	Attribute action: • 00b = deleted • 01b = modified - includes adding/removing/modifying attributes (e.g., Protocol records) for more than one of its Service Objects • 10b = reserved • 11b = added
reserved	130	301	reserved
LostTrap	431	1	This bit is set when the IOU had to discard a Trap without receiving a TrapRepress() for it. The bit is reset when the IOU receive a valid TrapRepress() with this bit set.

CA8-26: The DevMgt agent shall create the IO Controller Change notice as specified in Table 465 and issue the Trap from every port when an IOControllerProfile attribute record is added, deleted, or modified, unless the change is the result of a Slot Change reported by a Slot Status Change Notice.

A8.3.3.2.8 SERVICERECORD CHANGE NOTICE

The IOU uses the ServiceRecord Change notice to inform the DM that a ServiceRecord has been created, destroyed, or modified

Table 465 Notice 0x0011 DataDetails [ServiceRecord Change]

Field	Offset (bits)	Length (bits)	Description
IocGUID	0	64	IOC GUID from ServiceRecord attribute
ServiceObjectID	64	64	Service Object ID from ServiceRecord attribute
ACTION	128	2	Attribute action: • 00b = object deleted • 01b = modified (includes adding/removing/modifying Protocol records associated with the ServiceRecord). • 10b = reserved • 11b = object added
reserved	130	301	reserved
LostTrap	431	1	This bit is set when the IOU had to discard a Trap without receiving a TrapRepress() for it. The bit is reset when the IOU receive a valid TrapRepress() with this bit set.

CA8-27: The DevMgt agent shall create the ServiceRecord Change notice as specified in Table 465 and issue the Trap from every port when a ServiceRecord attribute is added, deleted, or modified, unless the change is result of a Slot Change reported by a Slot Change notice or an IOC change reported by an IO Controller Change notice.

A8.3.3.2.9 SLOT STATUS CHANGE NOTICE

The IOU uses the Slot Status Change Notice to inform the DM that an I/O module has been installed or removed.

Table 466 Notice 0x0018 DataDetails [Slot Status Change]

Field	Offset (bits)	Length (bits)	Description
SlotNumber	0	8	Slot Number of I/O module
us	8	4	Slot status - as per IOUnitInfo:SlotList.
reserved	12	419	reserved
LostTrap	431	1	This bit is set when the IOU had to discard a Trap without receiving a TrapRepress() for it. The bit is reset when the IOU receive a valid TrapRepress() with this bit set.

CA8-28: The DevMgt agent shall create the Slot Status Change notice as specified in Table 466 and issue the Trap from every port when its IOUnitInfo:SlotList changes.

A8.3.3.2.10 IOM REMOVAL NOTICE

The IOU uses the IomRemoval notice to inform the DM that a local action is requesting removal of an I/O module. This implies removal of one or more IOCs.

Table 467 Notice 0x0020 DataDetails [IomRemoval]

Field	Offset (bits)	Length (bits)	Description
SlotNumber	0	8	I/O module slot number
reserved	8	423	reserved
LostTrap	431	1	This bit is set when the IOU had to discard a Trap without receiving a TrapRepress() for it. The bit is reset when the IOU receive a valid TrapRepress() with this bit set.

oA8-3: The DevMgt agent shall create the IomRemoval notice as specified in Table 467 and issue the Trap from every port when an I/O module's SlotControlStatus:ModuleStatus.IOU_RTR bit transitions from 0 to 1. See A8.6 IOC Graceful Hot Removal on page 1618.

A8.3.3.2.11 DIAGSESSIONSTATE NOTICE

The IOU uses this Notice to inform the DM that the Diagnostic Session has changed (e.g., is ready for diagnostic testing). The DM also uses it in a DevMgtReport() to inform the diagnostic program whether it can proceed.

Table 468 Notice 0x0801 DataDetails [DiagSessionState]

Field	Offset (bits)	Length (bits)	Description
DiagToken	0	64	Specifies the Diagnostic Session
DiagSessionStatus	64	4	Diagnostic Readiness status as per the DiagSession attribute status defined in A8.3.3.17 DiagSession on page 1598.
reserved	68	363	reserved - set to zero
LostTrap	431	1	This bit is set when the IOU had to discard a Trap without receiving a TrapRepress() for it. The bit is reset when the IOU receive a valid TrapRepress() with this bit set.

See A8.5.2 Preparing for Diagnostic Tests on page 1617 for requirements on generating this Trap.

A8.3.3.2.12 DIAG SESSION VIOLATION NOTICE

The IOU uses this Notice to inform the DM of an invalid diagnostic request.

Table 469 Notice 0x0802 DataDetails [DiagSession Violation]

Field	Offset (bits)	Length (bits)	Description
Violation	0	8	Violation 0x00 - Test exceeds DiagSeverity level 0x01 - Service Object outside of scope 0x02 - IOC outside of scope 0x03 - I/O module outside of scope 0x04 - Service Object does not exist 0x05 - IOC does not exist 0x06 - I/O module does not exist
Method	8	8	Method used in MAD that caused the violation
AttributeID	16	16	AttributeID used in MAD that caused the violation
ViolatedPortGUID	32	64	Port GUID of the port receiving the violation
OffendingPortGid	96	128	Requestor port GID from the MAD that caused the violation. Zero indicates that the MAD did not contain a GRH.

Table 469 Notice 0x0802 DataDetails [DiagSession Violation] (Continued)

Field	Offset (bits)	Length (bits)	Description
OffendingLIDADDR	224	16	SLID used in MAD that caused the violation
reserved	240	16	reserved
QP	256	24	Source Queue Pair from the MAD that caused the violation
reserved	280	8	reserved
DiagToken	288	64	MADHeader:Access_Key from the DevMgt MAD that caused the violation.
Timestamp	352	48	Timestamp when violation was detected. The format of the timestamp is the number of milliseconds since 1/1/70 00:00am Zero indicates Timestamp not supported
reserved	400	31	reserved
LostTrap	431	1	This bit is set when the IOU had to discard a Trap without receiving a TrapRepress() for it. The bit is reset when the IOU receive a valid TrapRepress() with this bit set.

CA8-29: The DevMgt agent shall create the DiagSession Violation notice as specified in Table 469 and issue the Trap from every port when it detects a Diagnostic Session violation.

A Diagnostic Session violation is defined as receiving a DevMgtSet(TestDeviceOnce) or DevMgtSet(TestDeviceLoop) specifying a test device not listed in the Diagnostic Session's scope or specifying a test that violates the Diagnostic Session's DiagSeverity. See A8.3.3.17 DiagSession on page 1598 and A8.5.3 Invoking Diagnostic Tests on page 1617 for requirements on generating this Trap. Note that an invalid DiagToken is reported via the DiagToken Violation Notice on page 1551.

A8.3.3.3 INFORMINFO

The InformInfo attribute is specified in 13.4.8.3 InformInfo on page 739. It is used in a DevMgtSet() sent to the DM by interested parties to register for (subscribe to) DevMgt Traps.

The InformInfo attribute provides information for subscribing to a class manager for event forwarding. See 13.4.8.3 InformInfo on page 739 and 13.4.11 Event Forwarding on page 745.

A8.3.3.4 DA INFO

This attribute provides the address of the DA and general information about the DM/DA. Client platforms send a DevMgtGet(DAInfo) to the DM to learn the address of the Device Administrator associated with the DM. The client can also use this attribute to distinguish between multiple instances of the same manager and managers for different configuration groups.

Table 470 DAInfo Attribute

Component	Access	Offset (bits)	Length (bits)	Description
DA_GID	RO	0	128	GID of the Port where the clients sends DevAdm MADs
DA_LID	RO	128	32	LID of the Port where the clients sends DevAdm MADs
reserved	RO	160	8	reserved
DA_QPN	RO	168	24	Destination QP where the clients sends DevAdm MADs
reserved	RO	192	64	reserved
ConfigGroupID	RO	256	128	A 128-bit universally unique identifier (UUID) as defined by ISO/IEC 11578 that uniquely identifies the configuration group.
ConfigGroup-Name	RO	384	256	Null terminated UTF-8 string that specifies the name for the configuration group.
DevMgtVerHigh	RO	640	8	Indicates the highest Device Management ClassVersion that this manager supports
DevMgtVerLow	RO	648	8	Indicates the lowest Device Management ClassVersion that this manager supports
DevAdmVerHigh	RO	656	8	Indicates the highest Device Administration ClassVersion that this manager supports
DevAdmVerLow	RO	664	8	Indicates the lowest Device Administration ClassVersion that this manager supports
reserved	RO	672	864	reserved

The DA_GID, DA_LID, and DA_QPN specifies the address where a client platform sends DevAdm MADs.

The ConfigGroupID is a 128-bit UUID created by the configuration management application to identify the configuration group. When there are multiple managers that serve the same configuration group (i.e., distributed management application) they all report the same ConfigGroupID. Thus, a client platform can quickly determine which DM/DAs serve which configuration groups by sending a DevMgtGet(DAInfo) to each DM.

When a client sees multiple DMs with the same ConfigGroupID, it means that it will receive the same information from each one and thus only needs to query and subscribe to one of them.That is, each SA registration indicates a different path to the same logical DM (e.g., different port, P_Key, etc.).

The ConfigGroupName specifies a human readable name (if any) that a system administer assigned to the configuration group. The default value of all zeros indicates that a name has not been assigned.

The DevMgtVerHigh, DevMgtVerLow, DevAdmVerHigh, DevAdmVerLow specifies the highest and lowest versions of The DevMgt class and DevAdm class that the DM/DA supports.

A8.3.3.5 IOUNITINFO

A DM uses DevMgtGet(IOUintInfo) to learn detailed information about the IOU, such as the IOU vendor, the number of I/O subassembly modules and their physical status, the number of IOCs, the number of protocol records, service objects, and service records, the number of QPs, and the number of resource pools. An IORM uses this attribute to discover if the IOU contains an option ROM, which could provide I/O drivers.

There is one IOUnitInfo record and thus the DevMgt agent ignores the ComponentMask and RMPP header and returns a single DevMgtGetResp(IOUnitInfo) MAD.

Table 471 IOUnitInfo Attribute

Component	Access	Offset (bits)	Length (bits)	Description
reserved	RO	0	16	reserved - Change_ID moved to MAD header.
MaxSlots	RO	16	8	Number of actual slots in SlotList. Slots refer to I/O modules that reside behind the channel adapter and contain IOCs. Modules may be permanent or plugable. A value of zero indicates 256.
IsActivelyManaged	RO	24	1	Indicates if the IOU is actively managed. The IOU sets this bit to 1 when the IOU's Manager_Key (see KeyInfo attribute) is set to a non-zero value.
reserved	RO	25	6	reserved
OptionROM	RO	31	1	Indicates presence of Option ROM. 1 = Present; 0 = Absent.

Table 471 IOUnitInfo Attribute (Continued)

Component	Access	Offset (bits)	Length (bits)	Description
SlotList	RO	32	1024	A series of 4-bit nibbles with each representing an IOU subassembly. Each 4-bit nibble can take the following values: • 0x0 = module not installed or not available (i.e., not powered up) • 0x1 = present and available • 0x2-0xE = reserved • 0xF = slot does not exist Bits 7-4 of the first byte (lowest offset) represent slot 0, bits 3-0 represent slot 1, bits 7-4 of the second byte represent slot 2, bits 3-0 represent slot 3, and so on to slot 255.
IouVendorID	RO	1056	24	I/O unit vendor ID, IEEE format for Organization Unique ID (OUI). Bits are in canonical order.
reserved	RO	1080	8	reserved
ProductID	RO	1088	32	A number assigned by the vendor to identify the type of IOU.
IouVersion	RO	1120	16	A number assigned by the vendor to identify the IOU version.
IocCount[a]	RO	1136	16	Number of I/O controllers present and thus the number of IOControllerProfile Records that the DevMgt Agent will return if it receives an IOControllerProfile query with ComponentMask=0 (i.e., IocGUID=any, SlotNumber=any, etc.), see Section 8.3.3.6, "IOControllerProfile," on page 1561.
IouProtocolRecordCount[a]	RO	1152	16	Number of Protocol Records that the DevMgt Agent will return if it receives a Protocol query for IocGUID=0 (all other ComponentMask bits=0), see Section 8.3.3.8, "ProtocolRecord," on page 1568. This is the number of management protocols associated specifically with the IOU (rather than a particular IOC).
TotalProtocolRecordCount[a]	RO	1168	16	Total number of Protocol Records that the DevMgt Agent will return if it receives a Protocol query with ComponentMask=0 (IocGUID=any, ServiceName=any, etc.), see Section 8.3.3.8, "ProtocolRecord," on page 1568.
IouServiceRecordCount[a]	RO	1184	16	Number of IOU ServiceRecords that the DevMgt Agent will return if it receives a ServiceRecord query for IocGUID=0 (all other ComponentMask bits=0), see Section 8.3.3.7, "ServiceRecord," on page 1564. This is the number of management services associated specifically with the IOU (rather than a particular IOC).
ServiceRecordCount[a]	RO	1200	16	Number of ServiceRecords that the DevMgt Agent will return if it receives a ServiceRecord query with ComponentMask=0 (i.e., IocGUID=any, ServiceName=any, etc.), see Section 8.3.3.7, "ServiceRecord," on page 1564.
reserved	RO	1216	320	reserved

a. For the case where the actual count exceeds the maximum value of 0xFFFF, the DevMgt agent reports the maximum value. Thus, the value 0xFFFF indicates '*0xFFFF or more*'.

The previous DevMgt class version supported a one-to-one relationship between IOCs and subassembly slots. This version supports a many-to-one relationship. That is, an I/O subassembly module can contain multiple IOCs. The IOUnitInfo attribute contains information on the number of subassembly modules the I/O unit can support (IOUnitInfo:MaxSlots). The IOUnitInfo:SlotList has an entry for every possible “slot”. I/O module slots may be physical or logical and those modules can be removable or permanent. Each entry in the SlotList component shows whether the corresponding subassembly is present. An IOU that does not have physical subassemblies has the option of reporting one subassembly and associate all IOCs to it (preferred) or could report one subassembly for each IOC (legacy - as per class version 1).

SlotNumber of an I/O module refers to its logical position in the Slot list. IOCs are associated with slots but are primarily identified by their IocGUID. The SlotNumber helps the DM identify which IOCs are associated with a physical module, especially when the module is removed, replaced, or relocated. Note that replacement of a module could be with IOCs of the same type or a completely different type.

A8.3.3.6 IOCONTROLLERPROFILE

The IOControllerProfile attribute (as defined in Table 472 IOControllerProfile on page 1562) provides detailed information about an I/O controller. An I/O Controller represents a circuit or process on the IOU that provides I/O service. The IOControllerProfile attribute enables a host to identify the vendor of the IOC and load the appropriate vendor supplied I/O driver. Note that in order to allow an IOC to support multiple I/O protocols, as well as management protocols, the I/O protocol information has been removed from the IOControllerProfile (previous version) and made available via ProtocolRecord attributes. The information in ProtocolRecords enables the IORM to identify the appropriate generic I/O driver and the information in the IOControllerProfile attribute enables the host to identify the appropriate vender supplied I/O driver. Refer to Annex A1: I/O Infrastructure on page 1121 for driver matching rules.

There is one IOControllerProfile record for each IOC and the IocGUID uniquely identifies the IOC. This attribute provides details of each IOC such as its IocGUID, its vendor, product ID, product revision levels, and other information that is specific to a given IOC but common to all of the service objects provided by that IOC.

IOControllerProfile information aids a DM in allocating the IOU’s resources to various clients and the SlotNumber associates the IOC with a particular I/O subassembly for graceful hot removal. The IOControllerProfile also provides a common way for an IORM to determine the characteristics of IOCs.

Additional information about the controller is specified in ProductInfo, ProtocolRecord, and ServiceRecord attribute records associated with the IOC (i.e., records with that IOC's IocGUID). A ProtocolRecord specifies a protocol that the IOC supports. ServiceRecords identify instances of services and correlate services with their ServiceIDs so a client may establish a channel to communicate with the service object.

Table 472 IOControllerProfile

CMsk[a] (bit)	Component	Access	Offset (bits)	Length (bits)	Description
0	IocGUID	RO	0	64	An EUI-64 GUID used to uniquely identify the controller. This could be the same GIUD as the TCA's Node GUID if there is only one controller using that GUID.
1	VendorID	RO	64	24	I/O controller vendor ID, IEEE OUI format
2	SlotNumber	RO	88	8	Identifies which I/O module (Slot Number in the IOUnitInfo:SlotList) to which this IOC is associated.
-	IocDeviceID	RO	96	32	A number assigned by the IOC vendor to identify the type of controller. This can be used by an Operating System to select a device driver (see Annex A1: I/O Infrastructure on page 1121).
-	Device Version	RO	128	16	A number assigned by the IOC vendor to identify the device version (see A1.2.4.1 Matching an I/O Controller with an I/O Device Driver on page 1125).
-	reserved	RO	144	16	reserved
-	Subsystem VendorID	RO	160	24	IEEE OUI of the vendor of the module, if any, in which the I/O controller resides in IEEE format; otherwise zero (see A1.2.4.1 Matching an I/O Controller with an I/O Device Driver on page 1125).
-	reserved	RO	184	8	reserved
-	SubsystemID	RO	192	32	A number assigned by the subsystem vendor identifying the type of module where the controller resides (see A1.2.4.1 Matching an I/O Controller with an I/O Device Driver on page 1125).
-	IocServiceRecordCount[b]	RO	424	16	Number of **ServiceRecords** that the DevMgt Agent will return for a ServiceRecord query matching this IOC's IocGUID (all other ComponentMask bits=0), see A8.3.3.7 ServiceRecord on page 1564. This includes records for IOC I/O management objects as well as I/O service objects.

Table 472 IOControllerProfile (Continued)

CMsk[a] (bit)	Component	Access	Offset (bits)	Length (bits)	Description
-	IocProtocolCount[b]	RO	440	16	Number of **Protocol** Records that the DevMgt Agent will return for a ProtocolRecord query matching this IOC's IocGUID (all other ComponentMask bits=0), see A8.3.3.8 ProtocolRecord on page 1568. This includes all IOC related protocols (I/O protocols, management protocols, etc.)
-	IocServiceObjectCount[b]	RO	456	16	Number of **Service Objects** associated with this IOC. This includes all IOC related services (I/O service objects, management service objects, etc.)
-	reserved	RO	472	40	reserved
-	ID String	RO	512	512	UTF-8 encoded string for identifying the controller.
-	reserved	RO	1024	512	reserved

a. ComponentMask bit - This column indicates which bit in the ComponentMask is set when the initiator wants only records that match the corresponding component value in the query. Only components with CMsk bit numbers assigned can be specified.
b. For the case where the actual count exceeds the maximum value of 0xFFFF, the DevMgt agent reports the maximum value. Thus, the value 0xFFFF indicates '*0xFFFF or more*'.

There are different ways to request IOControllerProfiles. Three examples (by IocGUID, by SlotNumber, and a wildcard request that returns all IOControllerProfiles) are provided below. The requestor will receive one or more response packets containing zero or more records via the RMPP protocol.

- Wildcard Query

  For a wildcard query, the initiator sets the ComponentMask to 0x0000 in the DevMgtGet(IOControllerProfile). The DevMgt agent returns an IOControllerProfile for each IOC.

- Query by IocGUID

  For a query by IocGUID, the initiator sets the IocGUID in the DevMgtGet(IOControllerProfile) to the desired value and sets the ComponentMask to 0x0001. The DevMgt agent returns the IOControllerProfile for the specified IocGUID.

- Query by SlotNumber

  For a query by SlotNumber, the initiator sets ComponentMask in the DevMgtGet(IOControllerProfile) to 0x0004 (bit 2 of the ComponentMask set) and sets the SlotNumber component in the DevMgtGet(IOControllerProfile) attribute to the desired Slot Number. The DevMgt agent returns an IOControllerProfile for each IOC with the specified slot number.

The number of service objects associated with the IOC is specified in IocServiceObjectCountb. There are a number of ProtocolRecords and ServiceRecords associated with each IOC. The number of these records are specified in IocProtocolCountb and IocServiceRecordCount.

A8.3.3.7 SERVICERECORD

The IOU maintains the list of ServiceRecords. For each service object, there is one or more ServiceRecords that contains that service object's IocGUID+ServiceObjectID. That is, if the IOC supports multiple I/O protocols, then there will be multiple ServiceRecords per I/O service object (each with the same ServiceObjectID but a different service name). Additionally, there are ServiceRecords for I/O management service objects.

The combination of IocGUID+ServiceObjectID+ServiceName uniquely identifies a ServiceRecord.

There are two types of service objects (I/O service objects and I/O management service objects). An I/O service object is a port, through which a client can access I/O devices. An I/O management service object is a port, through which a management application can manage service objects and their I/O devices. The ObjectType component in the ServiceRecord attribute specifies if the service is an I/O service or an I/O management service.

Each Service Object is associated with an IOC. The ServiceRecord contains the IOC's IocGUID plus a ServiceObjectID that together uniquely identifies the service object. The IOU may have I/O management service objects that are not associated with a particular IOC, in which case the IocGUID in the ServiceRecord is set to zero.

An IOC has multiple I/O services objects anytime the IOC provides multiple sets of I/O resources for which the same I/O client has to use a different QP (or set of QPs) to access each set. For example, an IOC that supports the SCSI RDMA Protocol (SRP) would have a ServiceRecord for each SCSI target port. Additionally, if that IOC supports one or more management protocols (such as for a JBOD or RAID configuration program), the IOC would also have additional ServiceRecords for them.

The DM uses ServiceRecords to learn what service objects exists and determine how many resources (number and type of QPs) get consumed when a client uses the specified service object. This information is useful when the manager needs to establish the size of resource pools.

The IORM and I/O driver uses information from the ServiceRecord to identify the protocol (ServiceName) and channel characteristics for communicating with the specified service object including the ServiceID that the client uses in CM:REQ and/or CM:SIDR_REQ messages.

Table 473 ServiceRecord Attribute

CMsk[a] bit	Component	Access	Offset (bits)	Length (bits)	Description
0	IocGUID	RO	0	64	**IocGUID** - Identifies the IOC that provides the service. IOCs that have multiple service objects or support multiple protocols can have multiple records, each with the same IocGUID value. A value of zero designates the service is associated with the IOU and not a particular IOC.
1	ServiceObjectID	RO	64	64	**ServiceObjectID** - identifies the service object within the IOC's domain. ServiceRecords with the same IocGUID+ServiceObjectID refer to the same service object. Note that a service object that supports multiple protocols has multiple ServiceRecords (each with the same IocGUID+ServiceObjectID value).
2	ServiceName	RO	128	512	**Service Name** (protocol name)- UTF-8 encoded, null-terminated protocol-specific name of the service as per Annex A3 from the ProtocolRecord attribute that describes the protocol supported by a channel created using this record's ServiceID.
3	ObjectType	RO	640	8	Indicates the type of service object • 0x00 = unknown/other • 0x01 = I/O Service Object • 0x02 = I/O Management Object other values reserved
4	CMValidation	RO	648	1	Set to one to indicate that the IOU validates Client_Key as part of CM:REQ and CM:SIDR_REQ.When set to zero, it indicates that the DM must allocate a DefaultPool specifying this service object if it wants to assign the service object to a client.
5	SharedUD	RO	649	1	Set to one to indicate that UD QPs are used for multiple clients. This bit is set to zero if the service object uses a different UD QP for each client, does not use UD service, or only supports a single client and the UD QP is released when the client logs out.
-	reserved	RO	650	6	reserved
	ClientCountMax	RO	656	16	Number of clients that can simultaneously access this service. A value of 0xFFFF is interpreted as not limited by the implementation.
-	ServiceID	RO	672	64	**ServiceID** - The 64-bit Service ID that the Service Client uses in CM class MADs to connect with a QP for the particular service object using the specified I/O Protocol. Refer to Chapter 12 and Annex A3 for definition and usage of Service IDs.

Table 473 ServiceRecord Attribute (Continued)

CMsk[a] bit	Component	Access	Offset (bits)	Length (bits)	Description
-	RC Channels	RO	736	16	Maximum number of **Simultaneous RC Channels Supported** - a value of 0xFFFF means not limited by the implementation or the protocol. Any other value indicates the maximum number of RC QPs that the IOC allows for this ServiceID.
-	UC Channels	RO	752	16	Maximum number of **Simultaneous UC Channels Supported** - a value of 0xFFFF means not limited by the implementation or the protocol. Any other value indicates the maximum number of UC QPs that the IOC allows for this ServiceID.
-	UD Channels	RO	768	16	Maximum number of **Simultaneous UD QPs Supported** - a value of 0xFFFF means not limited by the implementation or the protocol. Any other value indicates the maximum number of UD QPs that the IOC allows for this ServiceID.
-	RD Channels	RO	784	16	Maximum number of **Simultaneous RD QPs Supported** - a value of 0xFFFF means not limited by the implementation or the protocol. Any other value indicates the maximum number of RD QPs that the IOC allows for this ServiceID.
-	Send Message Queue Depth	RO	800	16	Maximum number of outstanding requests - the number of request messages that a client can send before receiving an indication that they have been completed. If the implementation supports multiple channels per client, this is the value for the first command channel (unless specified otherwise by the I/O protocol).
-	reserved	RO	816	8	reserved
-	RDMA Read Queue Depth	RO	824	8	Maximum number of RDMA Read transactions that the service object can have outstanding (sent to client). Actual limit is established in CM connection establishment.
-	Send Message Size	RO	832	32	Maximum size of Send Messages in bytes that the service object can receive on this channel.
-	RDMA Transfer Size	RO	864	32	Maximum size of outbound RDMA transfers initiated by the IOC - in bytes.
-	reserved	RO	896	640	reserved

a. ComponentMask bit - This column indicates which bit in the ComponentMask is set when the initiator wants only records that match the corresponding component value in the query. Only components with CMsk bit numbers assigned can be specified.

An initiator can request particular ServiceRecords using the ComponentMask in the DevMgtGet(ServiceRecord). This limits the records returned to a specific subset matching the component values specified in the re-

quest. Thus, in the DevMgtGet(ServiceRecord), the requestor sets the corresponding ComponentMask bit to one (as specified by the CMsk column in Table 473: ServiceRecord Attribute) and specifies the component value in the ServiceRecord attribute. Only components with a CMsk bit number assigned can be specified and the DevMgt agent ignores component values for components without a CMsk bit number and for components whose ComponentMask bit is zero. The DevMgt agent returns all ServiceRecords that match the specified components and are associated with the requestor's Client_Key (see A8.4.2 Filtering Information on page 1608). Thus, when the ComponentMask is zero, the DevMgt agent returns all ServiceRecords that are associated with the Client_Key.

A8.3.3.7.1 CM VALIDATATION

When set, the CMValidation bit indicates that the IOU does not permit a client to use the service object until after it has validated that the client is authorized to use the service object and that the connection does not exceed the client's allotted resources as specified in the appropriate PlatformPoolRecord and ClientPoolRecord. That is, the service object requires that the client passes its Client_Key in the CM MAD's PrivateData component or as part of a login sequence prior to the service object allowing the client access to its service.

The requirement for setting CMValidation are

a) The client provides its Client_Key (such as in PrivateData of the CM:REQ and CM:SIDR_REQ messages) before the client is permitted to use to the service object, Service objects may set up a channel with the client and use that channel to convey the Client_Key, as long as the Client_Key validation is performed prior to permitting any other access.

b) The IOU validates the access rights against the ClientPoolRecord matching that Client_Key.

c) The QP is allocated from that client pool, except for UD QPs where the QP serves multiple clients (i.e., SharedUD bit is set in the corresponding ServiceRecord). In the case where SharedUD =1 and the QP does not already exist, the QP is allocated from the general pool.

A8.3.3.7.2 SHAREDUD

The SharedUD component indicates whether UD QPs are used for multiple clients. This bit is set to zero if the service object uses a different UD QP for each client, if the service object does not use UD transport service, or if the service only supports a single client and the UD QP is released when the client logs out.

Since one UD QP can service multiple clients, when SharedUD is set, the IOU allocates the QP from the general pool rather than from an individual

client pool. Thus, this bit lets the DM and supervisor know that it does not need to account for the UD service in the pool table QP allocations.

A8.3.3.8 PROTOCOLRECORD

The ProtocolRecord attribute provides the IORM with information it needs to match a generic driver to the service object.

The DevMgt Agent maintains the list of protocol records. The ProtocolRecord attribute contains the IOC's IocGUID. Each record describes a protocol supported by the IOC, and if the IOC supports multiple protocols, then there will be multiple Protocol records for that IOC. Protocol types include I/O protocols and I/O management protocols. Thus, there will be one ProtocolRecord attribute for each I/O protocol and each I/O management protocol that the IOC supports. The IOU may have one or more management protocols that are not associated with a particular IOC, in which case the IocGUID in the ProtocolRecord attribute is set to zero.

The combination of IocGUID+OrgID+Protocol components uniquely identifies a ProtocolRecord attribute.

Table 474 ProtocolRecord Attribute

CMsk[a] bit	Component	Access	Offset (bits)	Length (bits)	Description
0	IocGUID	RO	0	64	**IocGUID** - Identifies the IOC that this protocol is associated with. Entries with same IocGUID refer to same IOC. IOCs that support multiple protocols will have multiple protocol records, each with the same IocGUID value. A value of zero designates the IOU. {*i.e., IocGUID=0 for IOU management and utility protocols like ROM Repository*}.
1	Category	RO	64	4	**Protocol category** as per Annex A3.3.2.
-	reserved	RO	68	4	reserved
3	OrgID	RO	72	24	24-bit IEEE assigned **OUI** of the organization defining the protocol.
4	Protocol	RO	96	16	**Protocol ID** - as specified by organization identified in OrgID
5	ProtocolType	RO	112	8	Indicates the type of service object this protocol is for: • 0x00 = unknown/other • 0x01 = I/O Protocol • 0x02 = Management Protocol other values reserved
	reserved	RO	120	8	reserved

Table 474 ProtocolRecord Attribute (Continued)

CMsk[a] bit	Component	Access	Offset (bits)	Length (bits)	Description
7	ServiceName	RO	128	512	**Service Name** for the protocol as per Service Naming Conventions defined in Annex A3. UTF-8 encoded, null-terminated character string (e.g., “Console.IBTA”).
-	ProtocolVerLow	RO	640	16	**Protocol Version**[b] (**Low**) - This is a protocol-specific value that specifies the lowest protocol version supported by the IOC.
-	ProtocolVerHi	RO	656	16	**Protocol Version**[b.] **(High)** - This is a protocol-specific value that specifies the highest protocol version supported by the IOC.
-	ServiceRecordCount[c]	RO	672	16	Number of **ServiceRecords** that match IocGUID+ServiceName of this record
-	Operations	RO	688	16	Supported **operation types** of this protocol. A bit set to 1 for affirmation of supported capability. **Bit: Name; Description** 0: ST; Send Messages To IOCs 1: SF; Send Messages From IOCs 2: RT; RDMA Read Requests To IOCs 3: RF; RDMA Read Requests From IOCs 4: WT; RDMA Write Requests To IOCs 5: WF; RDMA Write Requests From IOCs 6: AT; Atomic Operations To IOCs 7: AF; Atomic Operations From IOCs 8-13 reserved 14: VSi; Vendor Specific operations to IOC 15: VSo; Vendor Specific operations from IOC
-	reserved	RO	704	832	reserved

a. ComponentMask bit - This column indicates which bit in the ComponentMask is set when the initiator wants only records that match the corresponding component value in the query. Only components with CMsk bit numbers assigned can be specified.
b. Protocol version is a numerically increasing value where the higher value indicates a later version. Format of this value is protocol specific.
c. For the case where the actual count exceeds the maximum value of 0xFFFF, the DevMgt agent reports the maximum value. Thus, the value 0xFFFF indicates ‘*0xFFFF or more*’.

An initiator can request particular Protocol records using the ComponentMask in the DevMgtGet(ProtocolRecord). This limits the records returned to a specific subset matching the component values specified in the request. Thus, in the DevMgtGet(ProtocolRecord), the initiator sets the appropriate ComponentMask bit to one (as specified by the CMsk column in Table 474: ProtocolRecord Attribute) and specifies the component value in the ProtocolRecord attribute. Only components with CMsk bit numbers assigned can be specified and the DevMgt agent ignores component values for components without a CMsk bit number assigned and for com-

ponents whose ComponentMask bit is zero. The DevMgt agent returns all Protocol records that match the specified components. Thus, when the ComponentMask is zero, the DevMgt agent returns all ProtocolRecords.

A8.3.3.9 SLOTCONTROLSTATUS

The SlotControlStatus attribute provides the DM with the module removal status for an I/O module and the means to control the removal state. This attribute is used for Graceful IOC Hot Plug.

An IOU that supports Graceful IOC Hot Plug (see A8.6: IOC Graceful Hot Removal) must support both Get(SlotControlStatus) and Set(SlotControlStatus). Each removable I/O module has its own SlotControlStatus attribute as per Table 475. A non-removable I/O module does not have a SlotControlStatus. Thus, the number of attribute records returned for a wildcard query for all slots is equal to the number of removable I/O modules (one attribute for each slot or a removable I/O module whether or not the slot is populated). An IOU that does not support Graceful IOC Hot Plug only needs to support Get(SlotControlStatus) and returns a MADHeader:Status of *NoMatchingRecord* in the GetResp().

Table 475 SlotControlStatus Attribute

CMsk[a] bit	Component	Access	Offset (bits)	Length (bits)	Description
0	SlotNumber	RO	0	8	Slot number of the I/O module
-	RemovalControl	RW	8	8	Specifies action as follows: • 0x00 = no change • 0x01 = Reset IOU_RTR and SW_RTR to zero • 0x02 = Set SW_RTR to one • 0x03 = Reset SW_CTR to zero • 0x04 = Set SW_CTR to one and reset IOU_RTR and SW_RTR to zero other values reserved For a DevMgtGetResp() in response to a DevMgt-Set(SlotControlStatus), this is the value from the DevMgtSet(). For a response to a DevMgtGet() this value is zero.

Table 475 SlotControlStatus Attribute (Continued)

CMsk[a] bit	Component	Access	Offset (bits)	Length (bits)	Description
2	ModuleStatus	RO	16	16	Indicates status of the IO module • bit 0: **Module_Present** - a 1 indicates that the module is present • bit 1: **IOU_RTR** - a 1 indicates that the IOU is requesting removal of the module. • bit 2: **SW_RTR** - a 1 indicates that DM has started the removal process for this module. • bit 3: **SW_CTR** - a 1 indicates that the I/O module may be removed. all other bits reserved
-	AttentionControl	R/W	32	2	Specifies a change to the Attention LED state as follows: • 00b = Turn Attention LED OFF • 01b = Turn Attention LED ON • 10b = Blink Attention LED (Module Identify) • 11b = no change to Attention LED state Note that local events may also cause a change to the Attention LED state. For a DevMgtGetResp() in response to a DevMgtSet(SlotControlStatus), this is the value from the DevMgtSet(). For a response to a DevMgtGet() this value is returned as 11b.
3	AttentionLED-State	RO	34	2	Indicates status of the Attention LED • 00b = OFF (neither the IOU nor the DM has indicated an attention event) • 01b = ON (either the IOU or the DM has indicated an attention event) • 10b = Blink (indicates Module Identify) • 11b = reserved
-	reserved	RO	36	28	reserved

a. ComponentMask bit - This column indicates which bit in the ComponentMask is set when the initiator wants only records that match the corresponding component value in the query. Only components with CMsk bit numbers assigned can be specified.

An initiator can request particular SlotControlStatus attributes by using the ComponentMask in the DevMgtGet(SlotControlStatus). This limits the records returned to a specific subset matching the component values specified in the request. For example, in the DevMgtGet(SlotControlStatus), the initiator sets the ComponentMask to 0x0001 (i.e., bit 0 =1) and specifies the slot number in the SlotControlStatus attribute to get the attribute for a specific slot. When the ComponentMask is zero, the DevMgt agent returns all SlotControlStatus records. DevMgt agent rejects a DevMgt**Set**(SlotControlStatus) if the ComponentMask is not 0x0001. That is, the DM can read all of the slots with a single request, but may only set one module per MAD.

A8.3.3.10 RESET

The reset attribute permits the DM to initiate an IOU or IOC reset by issuing a DevMgtSet(Reset) MAD. The reset is targeted the to the particular IOC specified by the IocGUID. The DM can target the reset to the entire IOU by setting IocGUID to zero. In this case, the DevMgt Agent resets the entire IOU.

A reset can be graceful or immediate. Graceful means that the IOC attempts to complete any I/O transactions in progress (i.e., empties its receive queues), if it cannot, or if the IOC continues to receive requests, the IOU terminates the reset and indicates that the reset failed. There are other conditions that may occur that also prevent a graceful reset from completing. They also cause the reset to fail. If so, the DM may need to signal an immediate reset. Immediate means the reset occurs independent of IOC state. IOU policy determines what actions the IOU takes when it receives a Reset. The ResetStatus component in the DevMgtGetResp(Reset) indicates the IOU's reaction to the Reset MAD.

Table 476 Reset Attribute

CMsk[a] (bit)	Component	Access	Offset (bits)	Length (bits)	Description
0	IocGUID	R/W	0	64	A value of zero specifies an IOU reset, otherwise specifies the IOC GUID of the IOC.
-	ResetType	R/W	64	1	Reset Type • 0b - Graceful Reset - Shutdown and Reset. Complete outstanding commands, close files etc., then reset. • 1b - Immediate Reset - Perform the reset immediately. In a DevMgtGetResp() to a DevMgtGet(Reset), this component is reserved and set to zero.
	ResetScope	R/W	65	2	Scope of reset 00 - default (IOU determines scope of reset) 01 - Reset only software 10 - Reset only hardware 11 - Reset both hardware and software
	reserved	RO	67	5	Reserved
	ResetStatus	RO	72	8	Reset Status • 0x00 - Not attempting to Reset • 0x01 - Reset in progress • 0x02 - Reset failed
	Reserved	RO	80	48	Reserved

a. ComponentMask bit - This column indicates which bit in the ComponentMask is set when the initiator wants only records that match the corresponding component value in the query. Only components with CMsk bit numbers assigned can be specified.

CA8-30: When the IOU receives a DevMgtSet(Reset) with IocGUID=0 and ResetType = Immediate, it shall immediately attempt a reset of the entire IOU.

CA8-31: When the IOU receives a DevMgtSet(Reset) with an IocGUID that matches one of its IOCs and ResetType = Immediate, it shall immediately attempt a reset of the specified IOC.

CA8-32: When the IOU receives a DevMgtSet(Reset) with an IocGUID = 0 and ResetType = Graceful, it shall only attempt the reset if all I/O transactions for all IOCs have ceased.

CA8-33: When the IOU receives a DevMgtSet(Reset) with an IocGUID that matches one of its IOCs and ResetType = Graceful, it shall only attempt a reset of the specified IOC if all I/O transactions for that IOC have ceased.

A8.3.3.11 PRODUCTINFO

The ProductInfo attribute provides the means for an IOU to provide product specific information about the IOU or an IOC which might be useful for asset management and other ancillary management.

Table 477 ProductInfo Attribute

CMsk[a] (bit)	Component	Access	Offset (bits)	Length (bits)	Description
0	IocGUID	RO	0	64	An EUI-64 GUID used to uniquely identify the controller. This could be the same GIUD as the TCA's Node GUID if there is only one controller using that GUID. A value of zero signifies the IOU, any other value signifies an IOC.
-	ProductData	RO	64	1472	A null-terminated TLV encoded component containing multiple packed elements of product specific information (see A8.3.3.11.2 ProductData on page 1575).

a. ComponentMask bit - This column indicates which bit in the ComponentMask is set when the initiator wants only records that match the corresponding component value in the query. Only components with CMsk bit numbers assigned can be specified.

A8.3.3.11.1 IOCGUID

Each ProductInfo record provides product specific information about the IOU or an IOC. When the IocGUID is zero, the record contains product information for the IOU. Otherwise, the record contains product information about the IOC specified by the IocGUID.

A8.3.3.11.2 PRODUCTDATA

The ProductData component provides supplemental information about the IOU or IOC that might help identify the IOU/IOC. The ProductData is a fixed-length variable-content field as follows.

- The ProductData component contains a variable number of variable length elements and is terminated with the null value (0x00). All bytes after the termination byte should also be null bytes.
- Elements in ProductData are in TLV format. The first byte is the Type, the second byte is the Length (number of bytes in the value string), and the remainder of the element is the value string, in UTF-8 format. Type codes are specified in Table 478 ProductData Elements on page 1575.
- An element with a Length of zero means that the value for that element is unknown or not set.

Elements that are not valid for a particular implementation should be excluded from the component. The choice of which elements are included is implementation policy. The recommended practice is to order elements by their Type value, lowest to highest.

Table 478 ProductData Elements

Type Value	Length Value	Description (All strings are in UTF-8 and the content of each element is vendor specific)
0x00	0x00	Marks end of elements - ignore remainder of component data following this Type code
0x01	variable	**Vendor Name String** This string provides the name of the vendor that manufactured the IOU/IOC.
0x02	variable	**Vendor Model/Type String** This string provides model and type information for the IOU/IOC.
0x03	variable	**Serial Number String** This string provides the serial number of the IOU/IOC.
0x04	variable	**Firmware/OS Vendor Name String** This string provides the name of the firmware vendor that manufactured the environment code (e.g., RTOS vendor.).
0x05	variable	**Firmware/OS Version String** This string provides the version of the firmware code.
0x06	variable	**CPU Vendor Name String** This string provides the name of the CPU vendor
0x07	variable	**CPU Vendor Version String** This string provides the CPU version, stepping, etc.

Table 478 ProductData Elements (Continued)

Type Value	Length Value	Description (All strings are in UTF-8 and the content of each element is vendor specific)
0x08	variable	**Platform Name String** This string provides the local name assigned the IOU/IOC - usually assigned by the System Administrator to identify the IOU/IOC by name.
0x09	variable	**Operating Environment Name String** This string provides the name of the Operating Environment (RTOS, OS, etc.).

A8.3.3.12 KEYINFO

The KeyInfo attribute is used by the DM to set the IOU's Device Management Manager_Key and protection properties.

Table 479 KeyInfo Attribute

Component	Access	Offset (bits)	Length (bits)	Description
Manager_Key	R/W	0	64	The 8-byte Manager_Key used in DevMgt MADs by valid DMs. A KeyInfo:Manager_Key value of 0 means the DevMgt agent does not perform a MADHeader:Access_Key check (see Table 480, "Manager_Key Check," on page 1578).
ProtectBits	R/W	64	2	See A8.3.3.12.3 Manager_Key Check on page 1578 for details.
reserved	R/W	66	14	reserved
LeasePeriod	R/W	80	16	Timer value used to indicate how long the ProtectBits are to remain non zero after a failed MADHeader:Access_Key check with KeyType=Manager. The value of the timer indicates the number of seconds for the lease period. With a 16 bit counter, the period can range from one second to approximately 18 hours. 0 shall mean infinite. See A8.3.3.12.5 Manager_Key Recovery on page 1580 for details.
Violations	R/W	96	16	Number of MADs that have been received at this IOU since power-on or reset that have been dropped due to a failed MADHeader:Access_Key check with KeyType = Manager or attempt to access an attribute that has a 'DM only' access level. Counts the number of DevMgt MADs that have been received by the DevMgt agent that contain an invalid MADHeader:Access_Key. The counter Increments until the count reaches all 1s and then must be set back to zero to re-enable incrementing. When the DM sets this component to 0x0000, the counter is reset to 0x0000 and counting resumes. Setting the counter to a value other than zero results the counter being left unchanged.
BackwardCompatibilityLevel		112	4	Sets the desired Backward Compatibility Level. See A8.3.1.1.2: Backward Compatibility Level.
reserved	RO	116	12	reserved

Table 479 KeyInfo Attribute (Continued)

Component	Access	Offset (bits)	Length (bits)	Description
V1ClientKey		128	64	Specifies the client pool table record that determines which service objects that a host using class version 1 MADs is authorized to see and use. See A8.3.1.1.2: Backward Compatibility Level.
reserved	RO	192	1344	reserved

A8.3.3.12.1 MANAGER_KEY GENERAL USE

The DM's key (Manager_Key) provides a separate level of authentication that helps protect against receipt of improper management request messages. There are two Key values that are compared with each other. One is the MADHeader:Access_Key located in the MAD header of every DevMgt MAD. The other is the KeyInfo:Manager_Key that is read and set via the KeyInfo attribute.

A DM sets an IOU's Manager_Key value via the DevMgtSet(KeyInfo) and the DevMgt agent uses that value to authenticate trusted sources by validating that the MADHeader:Access_Key in received MADs with KeyType = Manager match the KeyInfo:Manager_Key value.

Similar to the model used for other management classes (M_Key, B_Key, etc.), this model assumes that the fabric has some level of physical security. The value of MADHeader:KeyType component and the MAD's method and attribute determines how the DevMgt agent handles the MADHeader:Access_Key validation.

A DevMgt agent always sets the MADHeader:Access_Key to zero in all MADs it creates. For MADs received by the DevMgt agent, the MAD's Method, the KeyInfo:Manager_Key, and the KeyInfo:ProtectBits determine how the DevMgt agent handles the MADHeader:Access_Key validation (as per Table 480 Manager_Key Check on page 1578). If a key check fails, then the DevMgt agent silently drops the packet and increments the KeyInfo:KeyViolations counter. It then issues a MgrKey Violation trap.

CA8-34: The KeyInfo:KeyViolations component shall be incremented once, each time the DevMgt agent receives a MAD for which the MADHeader:Access_Key check was performed according to Table 480 Manager_Key Check on page 1578 and failed. However, the counter shall not be incremented if its value is all 1's.

CA8-35: The DevMgt agent shall set the KeyInfo:KeyViolations component to 0x0000 when it receives a valid DevMgtSet(KeyInfo) with KeyVio-

lations=0x0000. A KeyViolations value other than 0x0000 shall leave the counter unchanged.

A8.3.3.12.2 MANAGER_KEY ASSUMPTIONS

Assumptions for using the Manager_Key are:

1) To use the correct key for each IOU, the DM or a higher-level manager keeps track of the keys for the IOUs that it is managing.
2) If a backup DM exists, it shares the Manager_Key for ease of failover.
3) The DM sets the Manager_Key, the ProtectBits, and the LeasePeriod via the KeyInfo attribute with one DevMgtSet(KeyInfo) MAD. A successful completion of this assignment indicates to the DM that it has taken ownership of the IOU.

A8.3.3.12.3 MANAGER_KEY CHECK

Matching the MADHeader:AccessKey to KeyInfo:Manager_Key implies MADHeader:KeyType=DM.

The success and affect of the MADHeader:Access_Key validation check depends on the value of the KeyInfo:Manager_Key, KeyInfo:ProtectBits, and on the method and attribute contained in the incoming MAD. If the key check succeeds then the DevMgt agent responds to the MAD. If the key check fails, the DevMgt agent silently drops the MAD (i.e., does not process it and does not send a response).

CA8-36: When the DevMgt agent receives a MAD with MADHeader:KeyType=DM, it shall perform the authentication determined by the contents of KeyInfo:Manager_Key and the KeyInfo:ProtectBits as per the behaviors described in Table 480, “Manager_Key Check,” on page 1578.

Table 480 Manager_Key Check

KeyInfo Component Values		DevMgt Agent Actions
Manager_Key	ProtectBits	
zero	any	The MADHeader:Access_Key of the MAD shall not be checked when the KeyInfo:Manager_Key is zero. As a result, no authentication is performed.
non-zero	00b	• Any DevMgt MAD received shall succeed if MADHeader:Access_Key matches the KeyInfo:Manager_Key component of the IOU. • DevMgtTrapRepress(*) shall succeed for any value in the MADHeader:Access_Key • A DevMgtGet(*) shall succeed for any value in the MADHeader:Access_Key • A DevMgtGet(KeyInfo) shall return the IOU’s Manager_Key in DevMgtGetResp(KeyInfo) allowing any DM to learn the Manager_Key for the IOU. • A DevMgtSet(*) shall fail if MADHeader:Access_Key does not match the KeyInfo:Manager_Key component of the IOU.

Table 480 Manager_Key Check (Continued)

KeyInfo Component Values		DevMgt Agent Actions
Manager_Key	ProtectBits	
non-zero	01b	• Any DevMgt MAD received shall succeed if MADHeader:Access_Key matches the KeyInfo:Manager_Key component of the IOU. • DevMgtTrapRepress(*) shall succeed for any value in the MADHeader:Access_Key • A DevMgtGet(*) shall succeed for any value in the MADHeader:Access_Key • A DevMgtGetResp(KeyInfo) shall return the KeyInfo:Manager_Key value set to zero if MADHeader:Access_Key does not match the KeyInfo:Manager_Key. This prevents unauthorized entities from learning the Manager_Key of the IOU. • A DevMgtSet(*) shall fail if MADHeader:Access_Key does not match the KeyInfo:Manager_Key component of the IOU.
non-zero	10b	• Any DevMgt MAD received shall succeed if MADHeader:Access_Key matches the KeyInfo:Manager_Key component of the IOU • DevMgtTrapRepress(*) shall succeed for any value in the MADHeader:Access_Key • A DevMgtGet(*) for an attribute with RAL of 'A' or 'C' shall succeed for any value in the MADHeader:Access_Key • A DevMgtGet(*) for an attribute with RAL of 'M' shall fail if MADHeader:Access_Key does not match the KeyInfo:Manager_Key component of the IOU. • A DevMgtSet(*) shall fail if MADHeader: Manager_Key does not match the KeyInfo:Manager_Key component of the IOU.
non-zero	11b	• Any DevMgt MAD received shall succeed if MADHeader:Access_Key matches the KeyInfo:Manager_Key component of the IOU • DevMgtTrapRepress(*) shall succeed for any value in the MADHeader:Access_Key • A DevMgtGet(*) for any attribute shall fail if MADHeader:Access_Key does not match the KeyInfo:Manager_Key component of the IOU. • A DevMgtSet(*) shall fail if MADHeader: Manager_Key does not match the KeyInfo:Manager_Key component of the IOU.

CA8-37: When the DevMgt agent receives a validated (as per Chapter 13) DevMgtGet() or DevMgtSet() and the Manager_Key checking succeeds according to the rules specified in Table 480 Manager_Key Check on page 1578, then the DevMgt agent shall generate a DevMgtGetResp().

CA8-38: If Manager_Key check specified in Table 480, "Manager_Key Check," on page 1578 fails, the DevMgt agent shall:

1) Silently drop the MAD (i.e., do not send a response).
2) Increment KeyInfo:KeyViolations. Incrementing shall stop when the component reaches all 1s.
3) Send a KeyViolation trap on all ports.

When a DevMgt agent sends a MAD to the DM there is no need for the DM to do any type of key authentication. For simplicity and consistency, the MADHeader:Access_Key is set to zero for all MADs issued by the DevMgt agent.

CA8-39: The DevMgt agent shall set MADHeader:Access_Key to zero in all DevMgt MADs that it sends.

The DM does not check the MADHeader:Access_Key in any received DevMgt class MAD.

A8.3.3.12.4 MANAGER_KEY INITIALIZATION

CA8-40: At power-up, the KeyInfo:Manager_Key, KeyInfo:ProtectBits, and KeyInfo:LeasePeriod shall be set to zero if KeyInfo is not saved in NVRAM; otherwise, they shall be set to the values stored in NVRAM.

In the case that the IOU is reset or rebooted, the protection afforded by the Manager_Key needs to be preserved. It is desirable that the KeyInfo:Manager_Key, KeyInfo:ProtectBits, and KeyInfo:LeasePeriod survive the reset. It is only permissible to reset KeyInfo attribute components if the reset is complete, such that the P_Key tables are also cleared, because resetting the P_Key table removes clients' ability to access the IOU while it is vulnerable.

CA8-41: When the IOU is reset or rebooted, the KeyInfo:Manager_Key, KeyInfo:ProtectBits, and KeyInfo:LeasePeriod shall be preserved. The only exception is that if the IOU's P_Key tables are cleared and KeyInfo is not saved in NVRAM, then the KeyInfo:Manager_Key, KeyInfo:ProtectBits and KeyInfo:LeasePeriod may be set to zero.

Using a DevMgtSet(KeyInfo), the DM may assign the subsequent Manager_Key, ProtectBits, and LeasePeriod. Note that the DM should ensure that the lease period allows ample time for the DM to sweep the IOUs in its configuration group to prevent the lease from expiring.

A8.3.3.12.5 MANAGER_KEY RECOVERY

The Manager_Key lease period timer starts when the DevMgt agent receives a DevMgt class MAD whose MADHeader:Access_Key fails the Manager_Key check. At this time, the DevMgt agent sends a KeyViolation trap to the DM (that is, if the DM set its information in the trap components of the ClassPortInfo attribute). This trap serves as a request to the DM to refresh the lease period, which it can do by issuing any DevMgt MAD with a MADHeader:KeyType=DM and MADHeader:Access_Key that matches the IOU's KeyInfo:Manager_Key. The lease period timer is reset to the value contained in KeyInfo:LeasePeriod when the DevMgt agent receives any MAD with MADHeader:Access_Key that matches the KeyInfo:Manager_Key.

If the DevMgt agent fails to receive a MAD with a MADHeader:Access_Key that matches KeyInfo:Manager_Key, then the lease period expires - clearing the KeyInfo:ProtectBits to zero and allowing anyone to read (and then set) the KeyInfo:Manager_Key.

In the case when a IOU initializes using NVRAM (e.g. **IsManager_KeyNonVolatile**=1b) then Manager_Key, ProtectBits, LeasePe-

riod are set to the values the DevMgt agent had saved in local-persistent storage. The TrapLID for each port is reset to zero and the DevMgt agent waits for the DM to come around to set the port's TrapLid before it can send any traps. If the DevMgt agent receives a MAD that fails the Manager_Key check and the TrapLID is reset, then the DevMgt agent cannot determine the LID of the DM needed to send the trap. In this case, the IOU does not send the trap and the lease period timer could expire, causing eventual take over by a new DM.

With the DevMgtGet(KeyInfo), any DM can detect whether the Manager_Key is set (although hidden) based on the ProtectBits. If the ProtectBits are non-zero, the KeyInfo:Manager_Key is set and hidden. Otherwise, the returned KeyInfo:Manager_Key is the real one even if it is zero. Failure to get a response after some number of attempts is an indication that the KeyInfo:Manager_Key is set and KeyInfo:ProtectBits = 10b or 11b.

A8.3.3.12.6 LEASE PERIOD

A DM specifies the Lease Period by setting the contents of the KeyInfo:LeasePeriod component. It is intended to allow the Manager_Key protection to 'expire' if the DM inadvertently goes away without sharing the Manager_Key.

CA8-42: The lease period timer shall start counting down toward zero when the DevMgt agent receives a DevMgt MAD for which the Manager_Key check was performed according to Table 480, "Manager_Key Check," on page 1578 and failed. If the lease timer count is already underway, it shall not be interrupted by the arrival of that MAD.

Furthermore, for each port capable of sending DevMgt traps, the DevMgt agent sends a MgrKey Violation trap described in Table 458 Notice 0x0000 DataDetails [MgrKey Violation] on page 1548 to the DM indicating that the lease timer has started counting. In response to that trap, the DM may renew the Lease Period by setting the proper MADHeader:Access_Key in the DevMgtTrapRepress() or any other DevMgt MAD it sends to the IOU. If all DMs with the proper Manager_Key have gone away, then the Lease Period will expire, allowing another DM to take over.

CA8-43: The DevMgt agent shall cease counting down the lease period timer and shall reset it to the value contained in KeyInfo:LeasePeriod component when the DevMgt agent receives any MAD on any port with MADHeader:Access_Key that matches the KeyInfo:Manager_Key.

CA8-44: The DevMgt agent shall set its KeyInfo:ProtectBits to zero when its lease period counter expires.

When the lease period expires, clearing the ProtectBits allow any DM to read (and then set) the KeyInfo:Manager_Key.

CA8-45: When the KeyInfo:LeasePeriod is set to zero, the lease period timer shall never expire.

Whether there is an out-of-band mechanism to reset data protected with a lease period of zero is outside the scope of this specification.

A8.3.3.13 IOURESOURCEINFO

A DM uses DevMgtGet(IouResourceInfo) to learn detailed information about the IOU's configurable resources, such as size and capacity of platform pool table, size and capacity of client pool table, the number of QPs already assigned to platforms, and the number of available QPs.

There is one IouResourceInfo record and thus the DevMgt agent ignores the ComponentMask and RMPP header and returns a single DevMgtGetResp(IouResourceInfo) MAD.

Table 481 IouResourceInfo Attribute

Component	Access	Offset (bits)	Length (bits)	Description
PlatformPoolTableSize[a]	RO	0	16	Number of Platform Pools, thus it is the maximum number of PlatformPoolRecords the IOU can have
PlatformPoolRecordCount[a]	RO	16	16	Number of Platform Pools currently configured
ClientPoolTableSize[a]	RO	32	32	Number of Client Pools, thus it is the maximum number of ClientPoolRecords the IOU can have
ClientPoolRecordCount[a]	RO	64	32	Number of Client Pools currently configured
ServiceObjectMaxCount[a]	RO	96	32	Maximum number of service objects that can be specified in a PlatformPoolRecord and a ClientPoolRecord.
reserved	RO	128	8	reserved
NumFreeQPs	RO	136	24	Number of QPs still available - This value limits the number of QPs that the DM can allocate to a Platform Pool.
reserved	RO	160	8	reserved
TotalReservedQPs	RO	168	24	Number of QPs that have been reserved for platform pools - i.e., sum of PlatformQPmin for all Platform Pools
reserved	RO	192	8	reserved
TotalReservedQPsTarget	RO	200	24	Number of QPs that have been requested for platform pools - i.e., sum of PlatformQPminTarget for all Platform Pools
reserved	RO	224	8	reserved

Table 481 IouResourceInfo Attribute (Continued)

Component	Access	Offset (bits)	Length (bits)	Description
TotalQPsInUse	RO	232	24	Number of QPs consumed by all platforms - i.e., sum of ClientQPsInUse for all Client Pools
reserved	RO	256	8	reserved
TotalAdditionalQPs	RO	264	24	Number of additional QPs consumed by all platforms - i.e., sum of PlatformAdditionalQPs for all Platform Pools
reserved		288	7	reserved
NonVolatile	RO	295	1	Set to 1 to indicate that Pool Tables are persistently saved across power cycles.
ManagerUpdateLock	R/W	296	8	Set by the DM to coordinate updates Bit 0 - Supervisors prohibited from modifying ClientPoolRecords Bit 1 - Clients prohibited from consuming QPs other bits reserved
ManagerUpdateLease	R/W	304	16	Specifies the number of milliseconds from the time the manager sets this record until the ManagerUpdateLock reverts to zero. A lease value of zero means infinite.
reserved		320	16	reserved
MaxNumbDiagSessions	RO	336	16	Specifies the maximum number of concurrent diagnostic sessions that the IOU supports. A value of zero means the IOU does not support DIagnostic Sessions.
MaxNumbDiagObjects	RO	352	32	Specifies the maximum number of objects that can be listed in the DiagSession:ObjectList.
MaxClientPriority	RO	384	8	Specifies the number of priority levels the IOU supports by specifying the maximum priority value that can be specified for client priority in PlatformPoolRecords and ClientPoolRecords.
IsClientPriorityRetroactive	RO	392	1	Indicates if changes to ClientPoolRecord:ClientPriority impacts the priority of existing connections. 0 = only impacts priority of future connections 1 = impacts priority of existing as well as future connections
MaxPriorityChangeTime	RO	393	7	Indicates the maximum number of seconds that the IOU takes to change a QP's priority when ClientPoolRecord:ClientPriority changes. If IsClientPriorityRetroactive =0, then this component is undefined and set to zero.
	RO	400	1136	reserved

a. For the case where the actual count exceeds the maximum value, the DevMgt agent reports the maximum value. Thus, the value 0xFFFF in a 16-bit filed indicates '*0xFFFF or more*'.

A8.3.3.13.1 TABLE SIZES

The PlatformPoolTableSize and ClientPoolTableSize components respectively specify the maximum number of Platform Pool Table entries and Client Pool Table entries the IOU supports.

The PlatformPoolRecordCount and ClientPoolRecordCount components respectively specify the actual number of Platform Pool Table entries and Client Pool Table entries in use. Their values change when the DM creates or destroys Pool Table Records via the DevMgtSet(PlatformPoolRecord) and DevMgtSet(ClientPoolRecord) with Action = Create or Delete.

The number of pool table entries affects the ability of the DM to distribute resources. The number of platform pool table records affect the number of client platforms that may access the IOU.

CA8-46: An IOU shall support a minimum of 4 Platform Pool Records.

The number of client pool table records needed affect the number of clients that may access service objects. Ideally, there would be one record per service object per client platform. There must be at least one record per client platform and when there is a large number of service objects there needs to be at least one record for each service object.

CA8-47: An IOU shall support a number of Client Pool Table records equal to or greater than the number of Platform Pool Records or the number of Service Objects, which ever is greater.

ServiceObjectMaxCount specifies the maximum number of service objects that may be listed in a pool table entry. Ideally, that number is equal to the number of service objects. For the case where management creates a new service object for each client platform, the number of service objects necessary in a pool table record is significantly less than the total number of service objects. At a minimum, the DM needs the ability to list at least one service object per IOC.

CA8-48: An IOU shall support a ServiceObjectMaxCount equal to or greater than the number of IOCs.

A8.3.3.13.2 QP RESOURCES

The maximum number of QPs available can vary from one moment to the next depending on circumstances outside of Device Management. Additionally, the number of QPs in use is a dynamic value. The DM can calculate the maximum QPs available via adding the TotalReservedQPs, TotalAdditionalQPs, and the NumFreeQPs components, which represent a snapshot at the time the DM reads the IouResourceInfo attribute.

When there are sufficient QPs to fill a DevMgtSet(PlatformPoolRecord.PlatformQPminTarget) request, the IOU reserves the QPs for the platform and the TotalReservedQPs component will equal the TotalReservedQPsTarget. When there are not sufficient free QPs (i.e., NumFreeQPs = 0), then TotalReservedQPsTarget can be greater than TotalReservedQPs. As QPs become available (i.e., DM reduces QPmin of other pool table records or additional QPs are released), the DevMgt agent allocates those resources to platform pools that have a PlatformQPminTarget exceeding its PlatformQPmin. Thus, TotalReservedQPs will increment until it reaches TotalReservedQPsTarget.

The actual number of QPs in use (as reported in the TotalQPsInUse component) is different than the number of QPs reserved for platforms. This value represents the QPs that clients are actually consuming. The TotalAdditionalQPs component specifies the number of QPs that client platforms are consuming above their QPmin.

A8.3.3.13.3 UPDATE LOCK

The ManagerUpdateLock and ManagerUpdateLease components provide a means for the DM to lock out clients and supervisors while it is modifying pool table records. When the DM sets a non-zero value for the ManagerUpdateLock, the DevMgt Agent starts a timer for the time specified in ManagerUpdateLease. If the timer expires, the DevMgt Agent resets the ManagerUpdateLock to zero. This protects from the manager going away and leaving the IOU disabled. The DM cancels the lease by setting a ManagerUpdateLock value of zero. If the DM needs more time, it simply writes the IouResourceInfo attribute again, which restarts the timer with the new ManagerUpdateLease value.

A8.3.3.13.4 MAXCLIENTPRIORITY

MaxClientPriority specifies the number of priority levels the IOU supports by specifying the maximum priority value that may be specified for client priority in PlatformPoolRecords and in ClientPoolRecords. A value of zero indicates that the IOU supports one priority (i.e. the IOU does not support client priority). A value of “n” indicates the IOU supports “n+1” priorities, for example, a value of 3 indicates that the IOU supports the 4 priority levels 0 through 3.

A8.3.3.14 PLATFORMPOOLRECORD

The DM uses PlatformPoolRecords to create Supervisor_Keys and program the IOU with access privileges for individual platforms. Each record represents a client-platform, which is identified by a unique Supervisor_Key. The platform’s supervisor can read its PlatformPool-

Record to learn which service objects the platform is allowed to access as well as the resources that the platform's clients are permitted to consume.

Table 482 PlatformPoolRecord Attribute

CMsk[a] (bit)	Component	Access	Offset (bits)	Length (bits)	Description
0	Supervisor_Key	RO	0	64	A key that identifies the supervisor of the node permitted to access the service objects listed below.
-	ClientPoolCount	RO	64	32	The number of ClientPoolRecords with this Supervisor_Key
-	reserved	RO	96	32	reserved
-	Action	R/W	128	8	Specifies the action to be taken when this record is written. This component is returned with a value of zero in response to a DevMgtGet(). 0x00 - Update: Replace the record that matches this attribute's Supervisor_Key with this record. 0x01 - Create a new record 0x04 - Delete the PlatformPoolRecord with this Supervisor_Key and delete all ClientPoolRecords with this Supervisor_Key 0x07 - Delete all PlatformPoolRecords and all ClientPool-Records Any other value is an error - do not modify any record
-	ManagerUpdateLock	RO	136	8	ManagerUpdateLock value as defined in the IouResource-Info attribute. This component provides the means for the Supervisor to learn that the DM has asserted the global lock.
-	Non-Volatile	RO	144	1	Indicates if Pool Tables are preserved over power cycles.
-	SharedRecord	R/W	145	1	Specifies that this record is used by more than one platform.
-	reserved	RO	146	14	reserved
-	PlatformQoS	R/W	160	16	Indicates SL that the IOU is allowed to use with this platform' clients. Bit 0=SL0, Bit 1=SL1, ... Bit 15=SL15
-	reserved	RO	176	24	
-	PlatformQPmin	RO	200	24	Indicates the number of IOU QPs reserved for this platform.
-	reserved	RO	224	8	
-	PlatformQPminTarget	R/W	232	24	Specifies the desired QP minimum. The DM sets this value to change QPmin. The DevMgt Agent changes PlatformQP-min as soon as QPs become available.
-	reserved	RO	256	8	
-	PlatformQPmax	R/W	264	24	Specifies the maximum number of IOU QPs this platform is allowed to consume.
-	reserved	RO	288	8	

Table 482 PlatformPoolRecord Attribute (Continued)

CMsk[a] (bit)	Component	Access	Offset (bits)	Length (bits)	Description
-	PlatformQPsAllocated	RO	296	24	Indicates the number of IOU QPs that the supervisor has allocated to its client pools - This is the sum of ClientQPmin for all ClientPoolRecords with this Supervisor_Key
-	reserved	RO	320	8	
-	PlatformQPsAllocatedTarget	RO	328	24	Number of QPs that have been requested for this platform's client pools - it is the sum of ClientQPminTarget for all ClientPoolRecords with this Supervisor_Key.
-	reserved	RO	352	8	
-	PlatformQPsInUse	RO	360	24	Indicates the number of IOU QPs that are currently being used by this platform's clients - It is the sum of ClientQPsInUse for all ClientPoolRecords with this Supervisor_Key.
-	reserved	RO	384	8	
-	PlatformAdditionalQPs	RO	392	24	Indicates the number of QPs that the platform is consuming in excess of the number reserved for that platform (PlatformQPmin).This is the number of QPs borrowed from the general pool. - This value is calculated as the greater of zero or [*Sum {ClientAdditionalQPs} - PlatformUnallocatedQPs*], • where *PlatformUnallocatedQPs* = [PlatformQPmin - PlatformQPsAllocated], and *Sum {ClientAdditionalQPs}* is the total of [ClientQPsInUse-ClientQPmin] for ClientPoolRecords with this Supervisor_Key and ClientQPsInUse greater than ClientQPmin.
-	PlatformPriorityMin	R/W	416	8	Indicates the minimum priority that may be assigned to a client of this platform. Valid values range from 0 to PlatformPriorityMax, where zero is the lowest priority.
-	PlatformPriorityMax	R/W	424	8	Indicates the maximum priority that may be assigned to a client of this platform. Valid values range from PlatformPriorityMin to IouResourceInfo:MaxClientPriority.
-	reserved		432	16	
-	ServiceObjCount	R/W	448	32	Specifies the number of service objects listed in ServiceObjList
-	ServiceObjList	R/W	480	varies	List of service objects that the platform is allowed to access. Each service object is 128 bits [64-bit IouGUID+64bit ServiceObjectID]. The length of this field is 128 x n where n is ServiceObjCount.

a. ComponentMask bit - This column indicates which bit in the ComponentMask is set when the initiator wants only records that match the corresponding component value in the query. Only components with CMsk bit numbers assigned can be specified.

The DM can request particular PlatformPoolRecords by using the ComponentMask in the DevMgtGet(PlatformPoolRecord). This limits the records

returned to a specific subset matching the component values specified in the request. For example, in the DevMgtGet(PlatformPoolRecord), the manager sets the ComponentMask to 0x0001 (i.e., bit 0 =1) and specifies the Supervisor_Key in the PlatformPoolRecord attribute to get the record for a specific platform. When the ComponentMask is zero, the DevMgt agent returns all PlatformPoolRecords.

DevMgt agent rejects a DevMgt**Set**(PlatformPoolRecord) containing more than one attribute. Thus, the DM can read multiple records, but can only perform a single write per transaction. For a DevMgtSet(PlatformPool-Record), the DevMgt agent ignores the ComponentMask.

A8.3.3.14.1 SUPERVISOR KEY

The platform is designated by its Supervisor_Key. The DM creates a record by specifying a unique Supervisor_Key and setting the Action component to Create. If the Supervisor_Key already exists or there are no platform pools available, the DevMgt Agent rejects the set. Otherwise it creates a new record with that Supervisor_Key.

CA8-49: The DevMgt Agent shall reject a DevMgtSet(PlatformPool-Record) with MADHeader:KeyType other than Manager.

CA8-50: The DevMgt Agent shall reject to a DevMgtGet(PlatformPool-Record) with MADHeader:KeyType not equal to DM or Supervisor.

CA8-51: In response to a DevMgtGet(PlatformPoolRecord) with MAD-Header:KeyType = Supervisor, the DevMgt Agent shall only return the record with Supervisor_Key matching MADHeader:Access_Key.

A8.3.3.14.2 ACTION COMPONENT

The Action component specifies the action that the IOU takes when the DM performs the DevMgtSet(PlatformPoolRecord). The normal action is to replace or create a record. However there are times when the DM may need to delete a specific record or delete all records.

DevMgt agent ignores the ComponentMask in a DevMgt**Set**(Platform-PoolRecord) because the Action component specifies the scope.

CA8-52: The DevMgt Agent shall reject to a DevMgtSet(PlatformPool-Record) with Action = Update if the Supervisor_Key does not exist.

CA8-53: The DevMgt Agent shall reject to a DevMgtSet(PlatformPool-Record) with Action = Create if the Supervisor_Key already exists.

CA8-54: The DevMgt Agent shall reject to a DevMgtSet(PlatformPool-Record) with Action = Delete Record if the Supervisor_Key does not exist.

A8.3.3.14.3 MANAGER UPDATE

The ManagerUpdateLock informs the supervisor if the DM has locked the IOU's resources while the manager is modifying pool table entries. Set by the DM in the IouResourceInfo attribute ManagerUpdateLock component to coordinate updates.

CA8-55: If ManagerUpdateLock specifies that supervisors are prohibited from modifying ClientPoolRecords, then that DevMgt Agent shall either reject or defer a DevMgtSet(ClientPoolRecord) with KeyType = Supervisor.

A8.3.3.14.4 NON-VOLITILE

Informs the supervisor if the information that the supervisor configures into its ClientPoolRecords will be retained across power cycles.

A8.3.3.14.5 PLATFORMQOS

PlatformQoS specifies the SLs that the platform's clients are allowed to use when connecting with a service object. Thus, this component specifies the ClientQoS values that the supervisor may specify in a ClientPoolRecord.

A8.3.3.14.6 PLATFORM QPMIN, QPMAX, AND QPMINTARGET

Refer to Figure 318 Allocating Resource Pools on page 1607. The PlatformPoolRecord indicates the number of IOU QPs reserved for that platform (**PlatformQPmin**) and the maximum number of IOU QPs that the platform is allowed to consume (**PlatformQPmax**) for use in communication between the platform's clients and the IOU. Because there might not be enough available QPs to increase a platform's QP min, the DM sets **PlatformQPminTarget** to the desired value. If the QPs are available, the DevMgt Agent increases PlatformQPmin immediately, otherwise, the DevMgt Agent increases PlatformQPmin over time as QPs become available.

When the supervisor reduces PlatformQPminTarget below PlatformQPmin, the DevMgt Agent immediately reduces PlatformQPmin to match.

The IOU rejects a CM request that would cause the number of QPs used by all the clients with that same Supervisor_Key to exceed PlatformQPmax - see A8.4 Resource Allocation Framework on page 1606.

The attribute also provides a count of QPs that have been allocated to or consumed by the platform's clients (see Figure 320 Consuming QPs from Resource Pools on page 1613).

- **PlatformQPsAllocated** = sum of ClientQPmin for all ClientPoolRecords with this Supervisor_Key. This value is the total number of IOU QPs that the supervisor has successfully allocated to its client pools.
- **PlatfromQPsAllocatedTarget** = sum of ClientQPminTarget for all ClientPoolRecords with this Supervisor_Key. This value is the total number of QPs that the supervisor has attempted to allocate to its client pools. When PlatformQPsAllocated is less than PlatfromQPsAllocatedTarget, it means that there are insufficient QPs - either because the supervisor has specified more than its QPmin or because the some of its clients have consumed all of the platforms excess QPs.
- **PlatformQPsInUse** = sum of ClientQPsInUse for all ClientPoolRecords with this Supervisor_Key. This value indicates the number of IOU QPs that are actually being used by this platform's clients.
- **PlatformAdditionalQPs** - This value indicates number of additional IOU QPs consumed by this platform in excess of its PlatformQPmin. It is calculated as the greater of zero or [Sum {*ClientAdditionalQPs*} minus *PlatformUnallocatedQPs*], -- where PlatformUnallocatedQPs = [PlatformQPmin - PlatformQPsAllocated], and Sum {ClientAdditionalQPs} is the total of [ClientQPsInUse-ClientQPmin] for ClientPoolRecords with this Supervisor_Key and ClientQPsInUse greater than ClientQPmin. This value indicates number of additional IOU QPs consumed by this platform in excess of its PlatformQPmin.

A8.3.3.14.7 PLATFORMPRIORITYMIN & PLATFORMPRIORITYMAX

The PlatformPoolRecord specifies the minimum priority (PlatformPriorityMin) and maximum priority (PlatformPriorityMax) that may be assigned to any client of that platform. The values of these components limit the value of Client Priority that the supervisor may specify in it's ClientPoolRecords.

An administrator uses these components to set relative priorities for client platforms. For example, setting PlatformPriorityMax and PlatformPriorityMin to the same value assures that the platform operates at that priority level.

Valid values are 0-n, where zero is the lowest priority possible and n is the value specified in MaxClientPriority of the IouResourceInfo attribute.The higher the priority value, the greater the access to resources.

CA8-56: The DevMgt Agent **shall reject** a DevMgtSet(PlatformPoolRecord) with a PlatformPriorityMax that is greater than the MaxClientPriority of the IouResourceInfo attribute.

CA8-57: The DevMgt Agent **shall reject** a DevMgtSet(ClientPoolRecord) with a ClientPriority that is lower than the PlatformPriorityMin in the corresponding PlatformPoolRecord.

CA8-58: The DevMgt Agent **shall reject** a DevMgtSet(ClientPoolRecord) with a ClientPriority that is greater than the PlatformPriorityMax in the corresponding PlatformPoolRecord.

CA8-59: The DevMgt Agent shall reject a DevMgtSet(PlatformPoolRecord) with a PlatformPriorityMax that is lower than the PlatformPriorityMin.

The DM changing PlatformPriorityMax or PlatformPriorityMin does not change ClientPriority in ClientPoolRecords regardless of whether the ClientPriority is within the new range. Thus, if the DM wants to enforce the change, it needs to also modify the ClientPoolRecords. However, changes to ClientPoolRecords does not necessarily effect existing connections.

A8.3.3.14.8 SERVICE OBJECT LIST

The ServiceObjList is a list of service objects that the platform is allowed to access. Each service object is designated by two components: IocGUID + ServiceObjectID (see Figure 317: Service Object Tuple). The IocGUID specifies the IOC providing the service object and ServiceObjectID species the IOC specific service object. An IocGUID value of zero indicates an IOU service object.

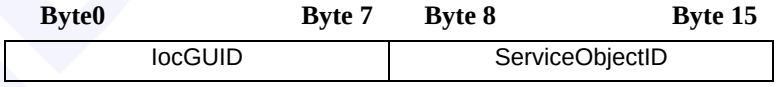


Figure 317 Service Object Tuple

The length of the ServiceObjList component is calculated as ServiceObjCount x 128 bits. The maximum size of the service object list is specified in the IouResourceInfo attribute ServiceObjectMaxCount.

A8.3.3.15 CLIENTPOOLRECORD

The DM uses ClientPoolRecords to create unique Client_Keys and the Supervisor uses the attribute to program the IOU with client access privileges for individual clients. See A8.4 Resource Allocation Framework on page 1606 for more information on how ClientPoolRecords are used.

Each record represents a client, which is identified by a unique Client_Key. The record also specifies the Supervisor_Key that associates the ClientPoolRecord to a particular PlatformPoolRecord. For each ClientPoolRecord, the platform's supervisor configures which service objects the client is allowed to access as well as the resources that the client is permitted to consume.

The platform's supervisor may only allocate resources (i.e., QPs, QoS, BW) permitted as per the respective PlatformPoolRecord.

Table 483 ClientPoolRecord Attribute

CMsk[a] (bit)	Component	Access	Offset (bits)	Length (bits)	Description
0	Client_Key	RW	0	64	A key that identifies the client permitted to access the service objects listed below.
1	Supervisor_Key	RW	64	64	A key that identifies the supervisor of this record and relates this record to its corresponding PlatformPool-Record.
-	Action	R/W	128	8	Specifies the action to be taken when this record is written. This component is returned with a value of zero in response to a DevMgtGet(). • 0x00 - Update: Replace the record that matches this attribute's Client_Key with this record. • 0x01 - Create a new record • 0x02 - Lock ClientPoolRecord with this Client_Key - make no other changes • 0x03 - Unlock ClientPoolRecord with this Client_Key - make no other changes • 0x04 - Delete the ClientPoolRecord with this Client_Key • 0x05 - Delete all ClientPoolRecords with this Supervisor_Key • 0x07 - Delete all ClientPoolRecords • Any other value is an error - do not modify any record Any Action other than 0x00 can only be performed by the DM
-	ManagerUpdateLock	RO	136	8	ManagerUpdateLock value as defined in the IouRe-sourceInfo attribute.
-	Non-Volatile	RO	144	1	Indicates if Pool Tables are preserved over power cycles.
5	LockedRecord	RO	145	1	When set to one, means that only the DM can modify the record. When set to zero the supervisor may modify the record. This value is set by the DM via the Action component.
6	DefaultPool	R/W	146	1	When set to one, means that this record describes the default pool for service objects listed in its Servi-ceObjList. When set to zero, means that this record describes a normal client pool.
-	reserved	RO	147	13	
-	ClientQoS	R/W	160	16	Indicates SL that the IOU is allowed to use with this client. Bit 0=SL0, Bit 1=SL1, ... Bit 15=Sl15
-	reserved	RO	176	24	

Table 483 ClientPoolRecord Attribute (Continued)

CMsk[a] (bit)	Component	Access	Offset (bits)	Length (bits)	Description
-	ClientQPmin	R/W	200	24	Indicates the number of IOU QPs reserved for this client.
-	reserved	RO	224	8	
-	ClientQPminTarget	R/W	232	24	Specifies the desired QP minimum. The supervisor sets this value to change QPmin. The DevMgt Agent changes ClientQPmin as soon as QPs become available.
-	reserved	RO	256	8	
-	ClientQPmax	R/W	264	24	Specifies the maximum number of IOU QPs this Client is allowed to consume.
-	reserved	RO	288	72	
-	ClientQPsInUse	RO	360	24	Indicates the actual number of IOU QPs this Client is currently consuming.
-	reserved	RO	384	32	
-	ClientPriority	R/W	416	8	Indicates the priority for this client. Valid values are limited by PlatformPriorityMin and PlatformPriorityMax of the corresponding PlatformPoolRecord. The higher the value the higher the priority.
-	reserved		424	24	
-	ServiceObjCount	R/W	448	32	Specifies the number of service objects listed in ServiceObjList
-	ServiceObjList	R/W		varies	List of service objects that the Client is allowed to access. Each service object is 128 bits [64-bit IouGUID+64bit ServiceObjectID]. The length of this field is 128 x n where n is ServiceObjCount.

a. ComponentMask bit - This column indicates which bit in the ComponentMask is set when the initiator wants only records that match the corresponding component value in the query. Only components with CMsk bit numbers assigned can be specified.

The DM or supervisor can request particular ClientPoolRecords by using the ComponentMask in the DevMgtGet(ClientPoolRecord). This limits the records returned to a specific subset matching the component values specified in the request. For example, in the DevMgtGet(ClientPoolRecord), the manager sets the ComponentMask to 0x0002 (i.e., bit 1 =1) and specifies the Supervisor_Key in the ClientPoolRecord attribute to get all the records for a specific platform. When the ComponentMask is zero, the DevMgt agent returns all ClientPoolRecords. A supervisor can only read its own ClientPoolRecords, so if it specifies a ComponentMask is zero, the DevMgt Agent returns its own ClientPoolRecords. A client cannot read any ClientPoolRecords.

DevMgt agent rejects a DevMgt**Set**(ClientPoolRecord) with more than one attribute. Thus, the DM or supervisor can read multiple records, but may only perform a single write. A supervisor can only write its own ClientPoolRecords. A client cannot write any ClientPoolRecords. For a DevMgtSet(PlatformPoolRecord), the DevMgt agent ignores the ComponentMask.

A8.3.3.15.1 CLIENT KEY

The Client's is designated by its Client_Key. The DM creates a record by specifying a unique Client_Key and setting the Action component to Create. If the Client_Key already exists, the DevMgt Agent rejects the set. Otherwise it creates a new record with that Client_Key.

CA8-60: The DevMgt Agent shall reject a DevMgtSet(ClientPoolRecord) with KeyType not equal to DM or Supervisor.

CA8-61: The DevMgt Agent shall reject a DevMgtGet(ClientPoolRecord) with MADHeader:KeyType not equal to DM or Supervisor.

A8.3.3.15.2 SUPERVISOR KEY

The Supervisor_Key correlates the ClientPoolRecord to a PlatformPoolRecord.

CA8-62: The DevMgt Agent shall reject a DevMgtSet(ClientPoolRecord) with KeyType = Supervisor if the attribute's Supervisor_Key does not match the MADHeader:Access_Key.

CA8-63: In response to a DevMgtGet(ClientPoolRecord) with MADHeader:KeyType = Supervisor, the DevMgt Agent shall only return records with Supervisor_Key matching MADHeader:Access_Key.

Note that by using component mask, in addition to requesting a particular ClientPoolRecord, the supervisor can request all of its own records.

A8.3.3.15.3 ACTION COMPONENT

The Action component specifies the action that the IOU takes when the DM performs the DevMgtSet(ClientPoolRecord). The normal action is to replace or create a record. However there are times when the DM may need to delete a specific record or delete all records. The DM is also able to lock the record. Only the DM can create, delete, lock, or unlock ClientPoolRecords.

DevMgt agent ignores the ComponentMask in a DevMgt**Set**(ClientPoolRecord) because the Action component specifies the scope.

CA8-64: The DevMgt Agent shall reject a DevMgtSet(ClientPoolRecord) with Action other than Update if KeyType is not Manager.

CA8-65: The DevMgt Agent shall reject a DevMgtSet(ClientPoolRecord) with Action = Update if the Client_Key does not exist.

CA8-66: The DevMgt Agent shall reject a DevMgtSet(ClientPoolRecord) with Action = Create if the Client_Key already exists.

CA8-67: The DevMgt Agent shall reject a DevMgtSet(ClientPoolRecord) with Action = Delete Record if the Client_Key does not exist.

CA8-68: The DevMgt Agent shall reject a DevMgtSet(ClientPoolRecord) if the Supervisor_Key does not exist, unless Action = Delete All.

A8.3.3.15.4 MANAGER UPDATE LOCK

The ManagerUpdateLock component is used to coordinate updates. It indicates if the DM has locked the IOU's resources while the manager is modifying pool table entries. The value is set by the DM via the IouResourceInfo attribute.

CA8-69: If ManagerUpdateLock specifies that supervisors are prohibited from modifying ClientPoolRecords, then that DevMgt Agent shall either reject or defer a DevMgtSet(ClientPoolRecord) with KeyType = Supervisor.

CA8-70: If ManagerUpdateLock specifies that clients are prohibited from consuming resources, then the IOU shall reject or defer CM:REQs for service objects.

A8.3.3.15.5 NON-VOLITILE

Informs the supervisor if the information that the supervisor configures into its ClientPoolRecords will be retained across power cycles.

A8.3.3.15.6 LOCKEDRECORD

Indicates if the record is locked and thus cannot be modified except by the DM. This value is set via the Action component. When not locked, the supervisor may modify the record.

The DM can lock ClientPoolRecords via the Action component. Locking a record prevents the supervisor from modifying the record. This is useful for Default Pools and for shared Platform Pools.

CA8-71: The DevMgt Agent shall reject a DevMgtSet(ClientPoolRecord) with KeyType = Supervisor if the record's LockedRecord bit is set.

A8.3.3.15.7 CLIENTQOS

Specifies the SLs that the Client is allowed to use when connecting with a service object.

CA8-72: The DevMgt Agent shall reject a DevMgtSet(ClientPoolRecord) if the ClientQoS is not a subset of PlatformQoS in the corresponding PlatformPoolRecord.

CA8-73: If the Primary SL or Alternate SL in a CM:REQ or CM:LAP is not a value specified in the ClientQoS component, then the IOU shall reject the CM request.

A8.3.3.15.8 CLIENT QPMIN, QPMAX, AND QPMINTARGET

Refer to Figure 318 Allocating Resource Pools on page 1607. The ClientPoolRecord specifies the number of IOU QPs reserved for that Client (ClientQPmin) and the maximum number of IOU QPs that the Client is allowed to consume (ClientQPmax) for use in communication between the clients and the IOU. Because the platform may have already consumed excess QPs, there might not be enough available QPs to increase a client's QP min. Thus, the supervisor sets ClientQPminTarget to the desired value. If the QPs are available, the DevMgt Agent increases ClientQPmin immediately, otherwise, the DevMgt Agent increase ClientQPmin over time as QPs become available.

When the supervisor reduces ClientQPminTarget, the DevMgt Agent immediately reduces ClientQPmin to match.

CA8-74: The DevMgt Agent shall not increase ClientQPmin if the sum of ClientQPmin for ClientPoolRecords with the same Supervisor_Key exceeds PlatformQPmin in the corresponding PlatformPoolRecord.

The DevMgt Agent rejects a CM request that would cause the number of QPs used by the client to exceed ClientQPmax - see A8.4 Resource Allocation Framework on page 1606.

A8.3.3.15.9 CLIENTPRIORITY

The ClientPoolRecord specifies the priority for that Client (ClientPriority). Valid values are limited by the corresponding PlatformPoolRecord (PlatformPriorityMin and PlatformPriorityMax), where the higher the value is higher priority.

Implementation of priority is not architected and is left as an implementation policy. However, a client with a higher priority must have equal or greater access to IOU resources than clients with lower priorities. For example, QPs for higher priority clients should have higher priority slots in the channel adapter's scheduling queue than QPs for lower priority clients. Thus, when the IOU completes a connection with a client, that QP/EEC assumes the priority assigned to that client via the corresponding ClientPoolRecord.

CA8-75: When the IOU creates a QP for a particular client, the IOU shall assign that QP the client priority from the corresponding ClientPool-Record.

CA8-76: If *IouResourceInfo:IsClientPriorityRetroactive* is one, then a change to ClientPoolRecord:ClientPriority must change the client priority for all QPs associated with that client within *IouResouceInfo:MaxPriority-ChangeTime* seconds.

CA8-77: The IOU shall assure that QPs with a given client priority are given equal or greater access to IOU resources than QPs with lower priority values.

A8.3.3.15.10 SERVICE OBJECT LIST

The ServiceObjList is a list of service objects that the client is allowed to access. Each service object is designated by two components: IocGUID + ServiceObjectID (see Figure 317 Service Object Tuple on page 1591).

The length of the ServiceObjList component is calculated as ServiceObj-Count x 128 bits. The maximum size of the service object list is specified in the IouResourceInfo attribute ServiceObjectMaxCount.

CA8-78: The DevMgt Agent shall reject a DevMgtSet(ClientPoolRecord) if the ServiceObjList is not a proper subset of the ServiceObjList in the corresponding PlatformPoolRecord.

A8.3.3.16 KEYCHANGE

The DM uses KeyChange attribute to change Client_Key and Supervisor_Key values. A successful DevMgtSet(KeyChange) means that the DevMgt Agent changed the Client_Key or the Supervisor_Key as specified in the attribute.

Table 484 KeyChange Attribute

Component	Access	Offset (bits)	Length (bits)	Description
Old_Key	W	0	64	Specifies the old Client_Key or Supervisor_Key value
New_Key	W	64	64	Specifies the new Client_Key or Supervisor_Key value
Action	W	128	8	Specifies the action to be taken when this record is written. • 0x01 - Change the Client_Key value in the ClientPoolRecord who's Client_Key = Old_Key to the value specified in New_Key. • 0x02 - Change the Supervisor_Key value in the PlatformPoolRecord who's Supervisor_Key = Old_Key to the value specified in New_Key and change Supervisor_Key value in all ClientPoolRecords who's Supervisor_Key = Old_Key to the value specified in New_Key Any other value is an error - do not modify any record
	RO	136	120	reserved

A8.3.3.17 DIAGSESSION

This attribute provides the means to establish, renew, or terminate a Diagnostic Session. A diagnostic application requests a Diagnostic Session by sending a DevMgtSet(DiagSession) to the DM and if the manager approves, the manager notifies the affected hosts and then sends a DevMgtSet(DiagSession) to the IOU. Renewing and terminating an established session work the same way. If the DiagSessionStatus in the response from the IOU indicates 'Device not Ready', then the IOU will send a trap when the status changes. If the manager accepts the session request, it responds to the diagnostic application with 'Device not Ready' so that it can inform the affected hosts and configure the IOU. It will then follow-up with a DevMgtReport() with the final status.

Table 485 DiagSession

CMsk[a] (bit)	Component	Access	Offset (bits)	Length (bits)	Description
-	IouGUID	R/W	0	64	The CA GUID of the IOU to be tested.

Table 485 DiagSession (Continued)

CMsk[a] (bit)	Component	Access	Offset (bits)	Length (bits)	Description
1	DiagToken	R/W	64	64	Diagnostic Token set by the DM. When this attribute is sent by the diagnostic program to establish a new session, it sets this value to zero in the Set() and the manager provides the actual value in the GetResp(). This component is ignored when Action = Terminate all existing diagnostic sessions.
-	DiagSeverity	R/W	128	4	Specifies the severity level of diagnostics that the diagnostic program may perform using this token: • 0x0 - Non-Intrusive - does not impact I/O performance • 0x1 - I/O performance reduced to 90% • 0x2 - I/O performance reduced to 80% • 0x3 - I/O performance reduced to 70% • 0x4 - I/O performance reduced to 60% • 0x5 - I/O performance reduced to 50% • 0x6 - I/O performance reduced to 40% • 0x7 - I/O performance reduced to 30% • 0x8 - I/O performance reduced to 20% • 0x9 - I/O performance reduced to 10% • 0xA - No I/O Service during diagnostics - connections remain • 0xF - No I/O Service - Connections terminated • other values reserved
-	DiagScope	R/W	132	4	Specifies the scope of the objects that can be affected by diagnostic testing using this token: 0x0 - Entire IOU: The diagnostics can affect all I/O objects of all IOCs. 0x1 - I/O Modules: The diagnostics can affect all I/O objects of the IOCs associated with the I/O modules listed in the ObjectList component. 0x2 - I/O Controllers: The diagnostics can affect all I/O objects of the IOCs listed in the ObjectList component. 0x3 - Service Objects: The diagnostics can affect the I/O objects listed in the ObjectList component. other values reserved
-	Action	R/W	136	4	Specifies the action that the DevMgt Agent takes when the DM sets this attribute: 0x0 - Establish a new diagnostic session 0x1 - Terminate all[b] existing sessions (if any) and establish a new diagnostic session 0x2 - Terminate existing diagnostic session (identified by DiagToken) 0x3 - Terminate all[b] existing diagnostic sessions 0x4 - Renew the lease for the Diagnostic Session identified by DiagToken. This may also establish new scope and diagnostic level. other values reserved

Table 485 DiagSession (Continued)

CMsk[a] (bit)	Component	Access	Offset (bits)	Length (bits)	Description
-	DiagSession-Status	RO	140	4	Indicates the state of the Diagnostic Session 0x0 = **Ready for diagnostic test** 0x1 = **Invalid DiagScope** - one or more of the objects listed is not present or does not exist 0x2 = **Device not ready** - More time is required to prepare one or more of the objects listed 0x4 = **Device not responding** - one or more of the objects listed has failed to prepare for diagnostic testing. 0x6 = **Device in use** - at least one of the objects listed is part of another DiagSession. 0x7 = **Unknown or Invalid DiagToken** - If Action = Terminate or Renew, DiagToken does not exist; Action = Establish, DiagToken already exists. 0x8 = **Diagnostics not supported** 0x9 = **Session Terminated** 0xF = **Request denied** other values reserved
-	LeasePeriod	R/W	144	16	Number of seconds remaining in the DiagSession
-	reserved		160	80	reserved
-	ObjectList-Count	R/W	240	16	Specifies the number of objects specified in objectList. When Diag-Scope is 'Entire IOU' then this value is zero. The max value allowed is reported in the IouResourceInfo attribute.
-	ObjectList	R/W	256	varies	Specifies the list of objects that can be impacted by a diagnostic test using this DiagToken. The content of this component is based on DiagScope and ObjectListCount: • Entire IOU: no data (this field size is zero) • I/O Modules: List of I/O Modules by slot number where each byte contains the slot number of an I/O module - thus the length of this component is n Bytes where n = ObjectList Count. • I/O Controllers: List of IocGUIDs - thus the length of this component is 8n Bytes where n = ObjectList Count. • Service Objects: List of IocGUID+ServiceObjectID tuples - thus the length of this component is 16n Bytes where n = ObjectListCount

a. ComponentMask bit - This column indicates which bit in the ComponentMask is set when the initiator wants only records that match the corresponding component value in the query. Only components with CMsk bit numbers assigned can be specified.
b. When sent to DevMgt Agent, 'all sessions' means any existing session. When sent by a diagnostic application to the manager, 'all sessions' means all sessions created by the source LID+QP.

The number of Diagnostic Sessions that an IOU supports is implementation dependant and reported in the IouResourceInfo attribute. The IOU rejects a DevMgtSet(DiagSession) that attempts to exceed the supported number of Diagnostic Sessions using MADHeader:Status of 'TableFull'. The IOU may also reject a DiagSession if the scope overlaps with an existing Diagnostic Session.

The diagnostic application uses the IouGUID component to indicate to the DM which IOU. If the DM permits the session, it then issues a DevMgtSet(DiagSession) to the IOU. If the IOU receives a DevMgtGet(DiagSession) or DevMgtSet(DiagSession) with an IouGUID that does not match the IOU's CA GUID, then the DevMgt Agent rejects the MAD with MADHeader:Status 2-4 = 7 (invalid value).

CA8-79: The DevMgt Agent shall reject a DevMgtGet(DiagSession) or DevMgtSet(DiagSession) with an IouGUID that does not match the IOU's CA GUID.

If a DevMgtGet(DIAGSession) requests a specific DIAG Session (i.e., specifies a specific DiagToken), then the IOU returns the attribute for that session. If the session does not exist, then the IOU returns an attribute with DiagSessionStatus = 0x7 "Unknown or Invalid DiagToken". If a DevMgtGet(DIAGSession) requests all DIAG Sessions (i.e., DiagToken CMsk bit not set), then the IOU returns all current DIAGSession Attributes (i.e., DIAG sessions that have not been terminated).

The LeasePeriod specifies the amount of time that the IOU keeps the session active. The time counts down from the time that DevMgtGetResp() is sent. If the lease timer expires, the session terminates. Note that the lease period granted by the manger may be different than what was requested by the diagnostic application. It the application needs more time, it can renew the lease before the end of the lease period. The lease period between the diagnostic application and the manager is to guard against the application disappearing before it terminates the session and the lease between the manager and the IOU is to protect against the manager disappearing. Thus, the manager may choose to use a different lease period with the IOU than it does with the diagnostic application.

CA8-80: The DevMgt Agent shall automatically terminate a DiagSession if the DM does not renew the lease within the time limit specified in DiagSession:LeasePeriod.

The diagnostic application uses the DiagToken in diagnostic DevMgt MADs that it sends to the IOU to specify the governing Diagnostic Session. The DevMgt Agent uses the DiagToken to authenticate the rights of the initiator to perform diagnostics by comparing the requested test to the permissions granted by the Diagnostic Session matching the DiagToken.

The IOU rejects a diagnostic test request that:

a) Specifies an invalid DiagToken

b) Exceeds the specified DiagSeverity level

c) Exceeds the specified DiagScope

CA8-81: The DevMgt Agent shall reject any DevMgtSet(TestDeviceOnce) and DevMgtSet(TestDeviceLoop) that does not specify a valid access key (i.e., valid DiagToken and MADHeader:KeyType=Diagnostic Token or the Manager_Key and MADHeader:KeyType=DM).

CA8-82: The DevMgt Agent shall reject any DevMgtSet(TestDeviceOnce) and DevMgtSet(TestDeviceLoop) that specifies a test object outside the scope of the Diagnostic Session matching the DiagToken.

CA8-83: The DevMgt Agent shall reject any DevMgtSet(TestDeviceOnce) and DevMgtSet(TestDeviceLoop) that specifies a test that exceeds the DiagSeverity level of the Diagnostic Session matching the DiagToken.

Note that if the IOU/IOC performs any automatic background diagnostics, those test should be inhibited while the device is included in an active Diagnostic Session. That is, while an object is listed in the scope of a Diagnostic Session, the DiagCode should only change as a result of a test explicitly run via the TestDeviceOnce or TestDeviceLoop commands.

A8.3.3.18 DIAGNOSTICTIMEOUT

This attribute provides the means to learn the maximum time required to perform the specified diagnostic. The result is based on the Test Type and Test Target. The R/W components indicate the information that is supplied in the DevMgtGet() to specify which test time,

Table 486 DiagnosticTimeout

Component	Access	Offset (bits)	Length (bits)	Description
MaxDiagTime	RO	0	32	Maximum time to finish the specified diagnostic operation in milliseconds
TestType	R/W	32	4	Specifies the type of test • 0x0=default • 0x2=vendor specific (VendorSpecific component provides additional details)
TestTarget-Type	R/W	36	4	Specifies the object type for the test • 0x0 = IOU • 0x1 = I/O module specified by ModuleSlot • 0x2 = IOC specified by IocGUID • 0x3 = I/O Object specified by IocGuid+ObjectID
reserved	-	40	16	reserved
ModuleSlot	R/W	56	8	I/O Module Slot Number
IocGUID	R/W	64	64	IOC GUID - Identifies the IOC for a test target of I/O object or IOC.
ObjectID	R/W	128	64	I/OC Object Identifier - Identifies the I/O object for a test target type of I/O object.

Table 486 DiagnosticTimeout (Continued)

Component	Access	Offset (bits)	Length (bits)	Description
reserved	-	192	64	reserved
VendorSpecific	R/W	256	varies	Additional information defined by the IOU vendor (not required for 'Default' test)

Note that the diagnostic framework provides for both '*default*' and '*vendor specific*' tests. Default refers to the test that is invoked by a diagnostic application that does not have vendor specific knowledge. Thus, the 'default' test permits the OS or a third party to perform a single diagnostic test (or set of tests) of the vendor's choosing on the specified target.

A8.3.3.19 TESTDEVICEONCE

This attribute allows the diagnostic program to initiate a diagnostic test that runs once.

Table 487 TestDeviceOnce

Component	Access	Offset (bits)	Length (bits)	Description
reserved	-	0	32	reserved
TestType	W	32	4	Specifies the type of test • 0x0=default • 0x2=vendor specific
TestTarget-Type	W	36	4	Specifies the object type for the test • 0x0 = IOU • 0x1 = I/O module specified by ModuleSlot • 0x2 = IOC specified by IocGUID • 0x3 = I/O Object specified by IocGuid+ObjectID
reserved	-	40	16	reserved
ModuleSlot	W	56	8	I/O Module Slot Number
IocGUID	W	64	64	IOC GUID - Identifies the IOC for a test target of I/O object or IOC.
ObjectID	W	128	64	I/OC Object Identifier - Identifies the I/O object for a test target type of I/O object.
reserved	-	192	64	reserved
VendorSpecific	W	256	varies	Additional information defined by the IOU vendor (not required for 'Default' test)

A8.3.3.20 TESTDEVICELOOP

This attribute allows the diagnostic program to initiate a diagnostic test that runs continuously.

This attribute provides several options for terminating the test. The behavior when the Action is set to 'run continuously' and the test encounters an error is not specified. That is, the test may continue with the next phase or might abort the current run and restart the diagnostic. In the event of an error, the DiagCode is set to indicate the error and is not changed until the test encounters another error or the test successfully completes a full loop without errors. Thus, the time that an error code remains available is indeterministic. The Action of 'Halt on Error' guarantees that an error code is not overwritten.

.

Table 488 TestDeviceLoop

Component	Access	Offset (bits)	Length (bits)	Description
reserved	-	0	32	reserved
TestType	W	32	4	Specifies the type of test • 0x0=default • 0x2=vendor specific
TestTarget-Type	W	36	4	Specifies the object type for the test • 0x0 = IOU • 0x1 = I/O module specified by ModuleSlot • 0x2 = IOC specified by IocGUID • 0x3 = I/O Object specified by IocGuid+ObjectID
Action	W	40	4	Specifies weather the test halts when it encounters an error or continues to loop. • 0x0 Continue to loop • 0x1 Halt on error • 0x2 Halt after current pass • 0x3 Abort
reserved	-	44	12	reserved
ModuleSlot	W	56	8	I/O Module Slot Number
IocGUID	W	64	64	IOC GUID - Identifies the IOC for a test target of I/O object or IOC.
ObjectID	W	128	64	I/OC Object Identifier - Identifies the I/O object for a test target type of I/O object.
reserved	-	192	64	reserved
VendorSpecific	R/W	256	varies	Additional information defined by the IOU vendor (not required for 'Default' test)

A8.3.3.21 DIAGCODE

This attribute provides the means to learn the result of the last diagnostic test performed on the specified object. Anyone may issue a DevMgtGet(DiagCode). Thus, a client may learn the status of the IOC without establishing a Diagnostic Session. In this case, the result could be from the IOC's 'power-on-test', the last diagnostic test performed as part of a Diagnostic Session, or background diagnostics automatically run by the IOU.

The R/W components indicate the information that is supplied in the DevMgtGet().

Table 489 DiagCode

Component	Access	Offset (bits)	Length (bits)	Description
DiagCode	RO	0	16	16-bit diagnostic code - a value of 0x0000 means "device operational" and all other values are IOU vendor-specific.
reserved	RO	16	16	reserved
TestType	R/W	32	4	Specifies the type of test • 0x0=default • 0x2=vendor specific
TestTarget-Type	R/W	36	4	Specifies the object type for the test • 0x0 = IOU • 0x1 = I/O module specified by ModuleSlot • 0x2 = IOC specified by IocGUID • 0x3 = I/O Object specified by IocGuid+ObjectID
reserved	-	40	16	reserved
ModuleSlot	R/W	56	8	I/O Module Slot Number
IocGUID	R/W	64	64	IOC GUID - Identifies the IOC for a test target of I/O object or IOC.
ObjectID	R/W	128	64	I/OC Object Identifier - Identifies the I/O object for a test target type of I/O object.
reserved	-	192	64	reserved
VendorSpecific	R/W	256	varies	Additional information defined by the IOU vendor (not required for 'Default' test)

A8.4 RESOURCE ALLOCATION FRAMEWORK

A8.4.1 QP ALLOCATION

The DM assigns resources to platforms, such as reserving a number of IOU QPs for a platform and limiting the maximum number of IOU QPs the platform may use by setting PlatformPoolRecords. The platform's supervisor assigns those resources to clients (within the bounds of the resources the DM assigned to the platform).

The IOU's QPs are represented as a general pool of QPs which consists of QPs reserved for client platforms, 'Free QPs' that are available on a 'first come' basis, and additional QPs used by platforms. The DM specifies a minimum (PlatformQPmin) and a maximum (PlatformQPmax) number of IOU QPs for each client platform as illustrated in Figure 318. QPmin represents the number of QPs reserved for that platform. When the DM increases a platform's QPmin, the number of Free QPs in the general pool decreases and the number reserved increase. Thus, a portion of the general pool becomes reserved for specific platforms. Remaining FREE QPs are available to platforms on a first come first served basis as long as the platform has not reached its maximum limit.

The total number of QPs in an IOU is typically static, but might change over time because non-I/O related processes consume and release QPs. This affects the number of FREE QPs, but does not change the number of QPs already reserved for platforms.

A supervisor allocates QPs from its platform pool to client pools by specifying a minimum (ClientQPmin) and a maximum (ClientQPmax) number of IOU QPs for each client. The ClientQPmin specifies QPs reserved for that client and the total number of QPs reserved for its clients cannot exceed the PlatformQPmin. Thus, QPs reserved for a platform (PlatformQPmin) are either Allocated and Unallocated. When the supervisor increases a client pool's QPmin value, the count of Platform Unallocated QPs decreases and the count of Platform Allocated QPs increases. A supervisor can only reserve up to the Platform's QPmin value for its clients (i.e., Sum of ClientQPmin <= PlatformQPmin).

As a client makes connections, the IOU uses QPs from the appropriate client pool. When a client has consumed all of the reserved QPs in its client pool (i.e., ClientQPmin), it may continue to consume additional QPs only if there are excess QPs available and the client pool has not reached its maximum limit. If the platform has un-allocated QPs (i.e., sum of clients QPmin is less than the platform's QPmin), then clients consume the platform's un-allocated QPs until they are exhausted. Once all the platform's un-allocated QPs are used, the clients may continue to consume available QPS from the general pool as long as the client's pool and the platform's pool have not reached their maximum limit. This means that the number

of reserved QPs plus additional QPs is less than QPmax both at a client level and at the platform level.

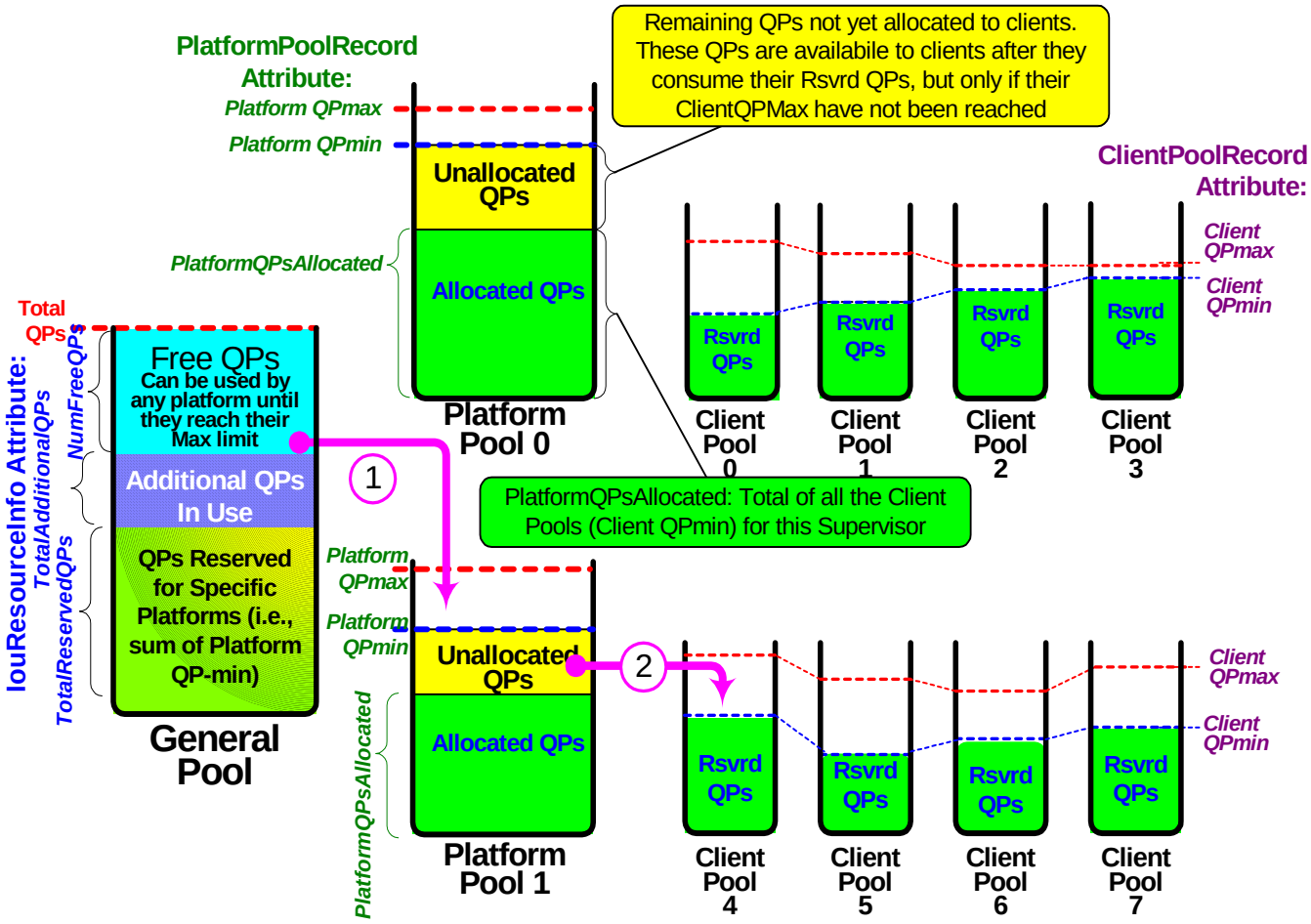


Figure 318 Allocating Resource Pools

When a QPmin or QPmax value is reduced, it does not affect any QPs in use until the client releases a connection and the QP is returned to the client pool. Thus, only if a platform has un-allocated QPs that have not been consumed by its clients, will reducing a platform's QPmin immediately return QPs to the FREE pool. Otherwise, QPs are returned to the Free pool when the QP is released by the client that has consumed more than its QPmin.

If the DM needs to increase QPmin for a platform, there needs to be Free-QPs in the general pool from which the additional QPs can be obtained.

The DM can reduce QPs allocated to other platforms (see previous paragraph) to increase the number of Free QPs.

Additionally, for a platform to consume more than its QPmin allotment, there has to be Free QPs available.

A8.4.2 FILTERING INFORMATION

ClientPoolRecords determine which records a particular client may see. PlatformPoolRecords determine which records a particular supervisor may see.

When the DevMgt agent receives a DevMgtGet() with a MADHeader:KeyType = Manager, it performs the Manager Key check as per A8.3.3.12.3 Manager_Key Check on page 1578. If the key check passes, the DevMgt Agent returns any records that match the query.

When the DevMgt agent receives a DevMgtGet() with a MADHeader:KeyType = Supervisor and the DevMgtGet() specifies an attribute that has a value of ‘S’ in the RAL column of Table 453 on page 1542, the DevMgt Agent uses the PlatformPoolRecord with Supervisor_Key = MADHeader:Access_Key and the PlatformPoolRecord with Supervisor_Key = 0 (if they exist). The combination of service objects in both records defines the IOCs and Service Objects for which the requester is authorized. For DevMgtGet(ServiceRecord), the DevMgt Agent only returns ServiceRecords that match the query and the service object appears in one of the two PlatformPoolRecords. For all other attributes, the DevMgt Agent returns only records that match the query and have an IocGUID listed in the combined lists.

Likewise, When the DevMgt agent receives a DevMgtSet() with a MADHeader:KeyType = Supervisor and the DevMgtSet() specifies an attribute that has a value of ‘S’ in the WAL column, the DevMgt agent does not allow the write unless the attribute matches one of the platform’s service objects.

CA8-84: If the DevMgt agent receives a DevMgt MAD with a MADHeader:KeyType = Supervisor, the DevMgtGet() specifies an attribute that has a value of ‘S’ in the RAL column of Table 453 on page 1542, and there is no PlatformPoolRecord with Supervisor_Key = MADHeader:Access_Key, then the DevMgt agent shall send a Supervisor Key Violation Trap.

CA8-85: If the DevMgt agent receives a DevMgt MAD with a MADHeader:KeyType = Supervisor, the MAD specifies an attribute that does not have a value of ‘S’ or ‘A’ in the respective RAL or WAL column of Table 453 on page 1542, then the DevMgt agent shall reject the MAD.

CA8-86: If the DevMgt agent receives a DevMgt MAD with a MADHeader:KeyType = Supervisor, the DevMgtGet() specifies an attribute that has a value of 'S' in the respective RAL or WAL column of Table 453 on page 1542, the DevMgt Agent shall only return records that match service objects listed in the PlatformPoolRecord with Supervisor_Key = MADHeader:Access_Key or listed in the PlatformPoolRecord with Supervisor_Key = 0.

When the DevMgt agent receives a DevMgtGet() with a MADHeader:KeyType = Client and the DevMgtGet() specifies an attribute that has a value of 'C' in the RAL column of Table 453 on page 1542, the DevMgt Agent uses the ClientPoolRecord with Client_Key = MADHeader:Access_Key (if it exists). The service objects listed in this record defines the IOCs and Service Objects for which the requester is authorized. For DevMgtGet(ServiceRecord), the DevMgt Agent only returns ServiceRecords that match the query and the service object appears in the ClientPoolRecord:ServiceObjList. For all other attributes, the DevMgt Agent returns only records that match the query and have an IocGUID listed in the ClientPoolRecord:ServiceObjList.

Likewise, when the DevMgt agent receives a DevMgtSet() with a MADHeader:KeyType = Supervisor and the DevMgtSet() specifies an attribute that has a value of 'C' in the WAL column, the DevMgt agent does not allow the write unless the attribute matches one of the client's service objects. Currently, there are no attributes that clients may modify.

CA8-87: If the DevMgt agent receives a DevMgt MAD with a MADHeader:KeyType = Client, the DevMgtGet() specifies an attribute that has a value of 'C' in the respective RAL or WAL column of Table 453 on page 1542, and there is no ClientPoolRecord with Client_Key = MADHeader:Access_Key, then the DevMgt agent shall send a Client Key Violation Trap.

CA8-88: If the DevMgt agent receives a DevMgt MAD with a MADHeader:KeyType = Client, the MAD specifies an attribute that does not have a value of 'C' or 'A' in the respective RAL or WAL column of Table 453 on page 1542, then the DevMgt agent shall reject the MAD.

CA8-89: If the DevMgt agent receives a DevMgt MAD with a MADHeader:KeyType = Client, the MAD specifies an attribute that has a value of 'C' in the respective RAL or WAL column of Table 453 on page 1542, the DevMgt Agent shall only return records that match service objects listed in the ClientPoolRecord with Client_Key = MADHeader:Access_Key.

A8.4.3 RESTRICTING ACCESS

The IOU must only permit authorized clients to access I/O objects. Authorized client means that the service object is listed in the ServiceObjList of the corresponding ClientPoolRecord. Thus, the service object must ascertain the Client_Key before it allows the client to access its service. In addition, the IOU needs to know which client is consuming the QP so it can track the number of QPs that each client is using and thus limit the resources that each client consumes.

CM MADs do not contain a Client_Key component. However, many I/O protocols require some form of validation, where client information is passed to the service object for validation. For SCSI this is the LOGIN, and the equivalent of the DevMgt Client_Key is the SCSI Initiator ID. For SRP, the Initiator ID is passed in the PrivateData field of CM MADs. Because I/O protocols already have architected ways of passing this type of information, this annex does not specify how the information is passed.

The CMValidation bit in the ServiceRecord identifies if the ServiceObject does validate the Client_Key when the client creates a connection.

- For service objects that validate clients (i.e., the CMValidation bit in the corresponding ServiceRecord is set), the QP is allocated from the client's pool, if the pool exists, but only if the supervisor has configured the pool to include that service object.
- For service objects that do not validate clients (i.e., the CMValidation bit in the corresponding ServiceRecord is not set), the QP is allocated from a default pool, if a default pool exists which specifies that service object. If the DM omits a service object that has CMValidation=0 from all Default Pool's ServiceObjList, then no one is allowed to access that service object.

Thus, when completing a connection, the IOC validates the access rights using the Client_Key. If the ServiceObject's CMValidation = 0, the then the IOU uses Client_Key = 0. If the Client_Key is valid (i.e., a ClientPoolRecord exists with that Client_Key), the QP is allocated from the client pool. Figure 319 Connection Approval Decision Tree on page 1611 illustrates the decision process.

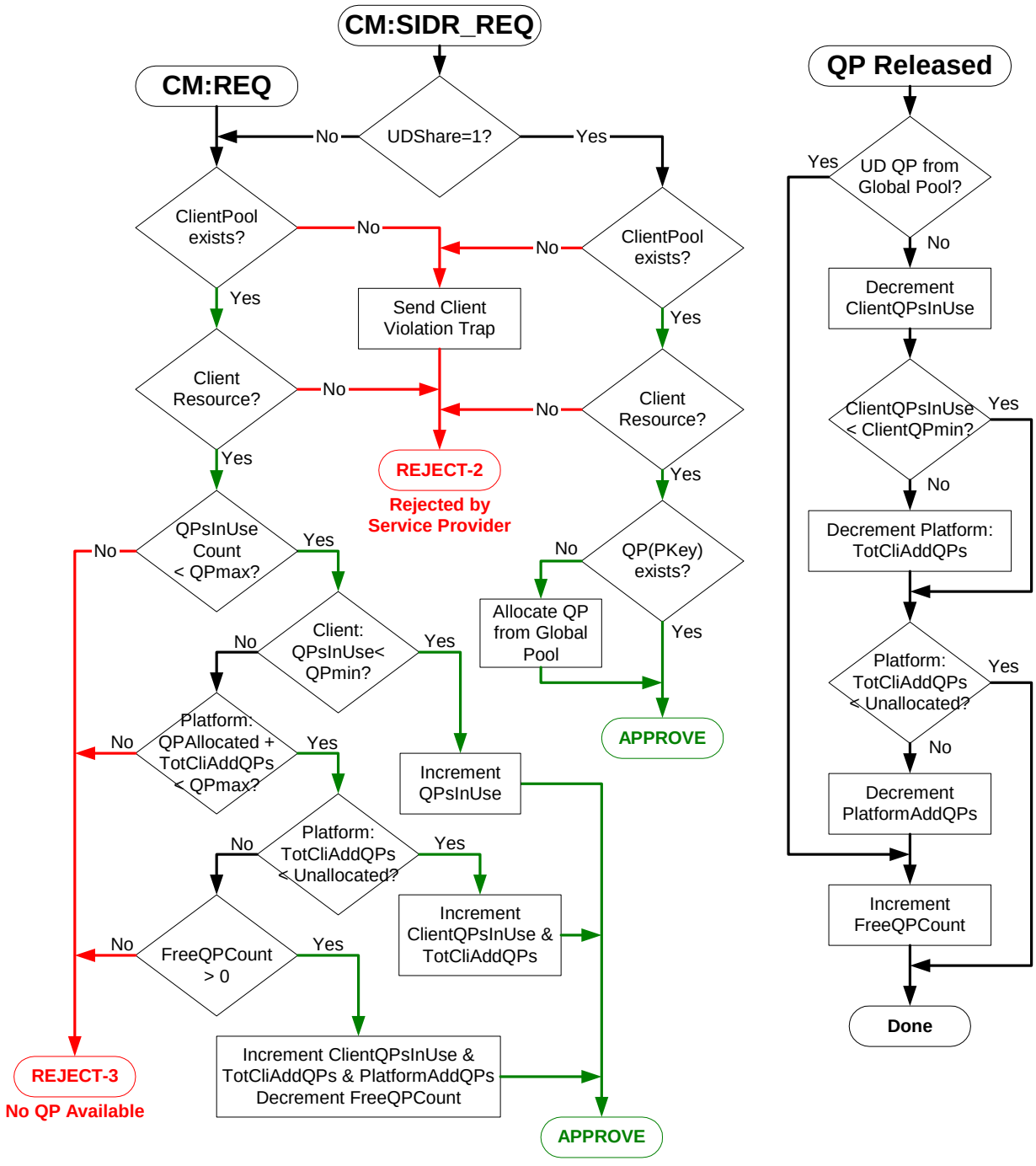


Figure 319 Connection Approval Decision Tree

The IOC shall not allow access to a service object until it validates the source's Client_Key matches a Client_Key in a ClientPoolRecord containing the service object's IocGUID and ServiceObjectID.

For protocols passing Client_Key in the CM PrivateData field, the validation is done before the QP is allocated, otherwise it is done when the client passes the Client_Key to the service object. For UD transport service, the term 'connection' means the ability of the client to access the service.

CA8-90: The service object shall inhibit service over a connection until the client provides a valid Client_Key (i.e., matches a Client_Key in a ClientPoolRecord) and the corresponding ClientPoolRecord lists the target service object in its ServiceObjList.

CA8-91: The IOU shall reject the connection and send a Client Validation Trap if the client passes a Client_Key that does not match a Client_Key in any ClientPoolRecord.

CA8-92: The IOU shall reject the connection if the target service object is not in the ServiceObjList of the ClientPoolRecord corresponding to the Client_Key passed to the service object.

CA8-93: The IOC shall reject the connection if there are no QPs available in appropriate Client Pool. That is, if the client has exceeded ClientQPmax, the platform has exceeded PlatformQPmax, or there are no free QPs in the general QP pool.

CA8-94: If the service object uses a UD protocol where each client uses a different UD QP (i.e., SharedUD not set in the ServiceRecord), then the IOC shall reject the CM:SIDR_REQ if there are no QPs available in appropriate Client Pool.

A8.4.4 CONSUMING QPS

The PlatformPoolRecord and the ClientPoolRecord specifies the minimum and maximum number of IOU QPs that a platform and client may consume respectively. These values are related as illustrated in Figure 320.

As a client makes connections, it uses QPs from it's client pool. When a client has consumed all of the reserved QPs in its client pool (i.e., QPmin), it may continue to consume additional QPs only if there are excess QPs available and the client pool has not reached its maximum limit.

- If the platform has un-allocated QPs (i.e., sum of clients QPmin is less than the platform's QPmin), then clients consume the platform's un-allocated QPs until they are exhausted.

- Once all the platform's un-allocated QPs are used, the clients may continue to consume available QPs from the general pool as long as the platform has not reached its maximum limit. This means that the number of reserved QPs plus additional QPs is less than QPmax both at the client level and at the platform level.

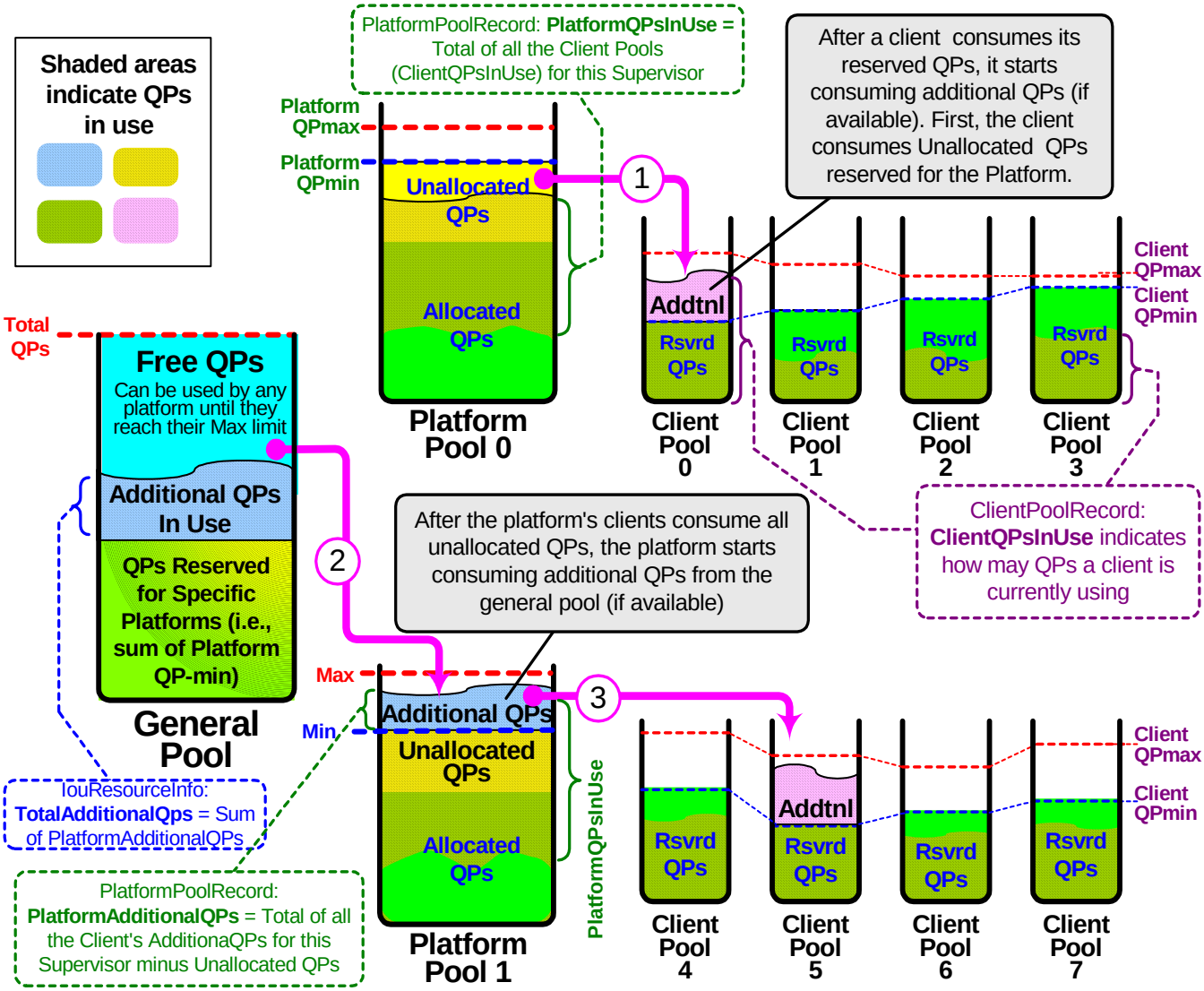


Figure 320 Consuming QPs from Resource Pools

When the client or service object terminates a connection, the IOU returns the freed QP to the appropriate pool as follows.

1) The DevMgt Agent decrements the In-Use count for that client pool.

2) If the number of QPs in-use in that client pool is now less than the client QPmin, then the freed QP remains allocated that client pool for future use by that client.

3) If the number of QPs in-use in that client pool is greater than or equal to the client QPmin, then the freed QP is returned to the platform pool as follows.

   a) If the total number of additional QPs in the platform's client pools plus the number of allocated QPs (i.e., sum of client QPmin) is less than the platform's QPmin, then the freed QP remains allocated the platform pool for future connections by any of the platform's clients.

   b) If the total number of additional QPs in the platform's client pools plus the number of allocated QPs (i.e., sum of its client QPmin) is greater than or equal to the platform's QPmin, then the freed QP returns to the general pool for use by all clients.

A8.5 DEVICE DIAGNOSTIC FRAMEWORK

Device diagnostics allows the identification of faults in devices behind the target channel adapter. As such, it complements other sections of this specification that describe how problems at the fabric and node level may be identified and isolated.

It is not the intent of this annex to define a set of white-box, technology-specific diagnostic tests. Rather, the intent is to allow initiation of a vendor-supplied test sequence (referred to as the default test), for which the expected outcome would be either success or failure. The DiagCode format, however, allows flexibility for the a vendor specific diagnostic program to run specific tests and for the vendor to provide specific, coded information about the test results. To this end, most diagnostic attributes contain a VendorSpecific component that allows the diagnostic program to specify additional information and allows the IOU to return additional information.

The device diagnostic framework is intended to support tests within an active fabric. It is versatile enough, however, to accommodate vendor-unique approaches that may include retrieval of power-on data. It should be noted that some, and perhaps most, devices may not permit simultaneous use of I/O transaction messages and diagnostics. Unless data is flushed from internal buffers, for example, corruption or loss of user data might occur. Thus, an IOU may reject a diagnostic request if the device is not in an appropriate state. Further, it is expected that the diagnostics tests would require setting the device to an initial, known state. For that reason, provision is be made to put the device into a "ready" state prior to test, which could cause I/O transactions to be held off. This may, in turn,

cause established connections to time out, and other management notices to be sent.

To prevent an I/O client from being surprised because of side effects from performing diagnostics, the diagnostic application must first establish a Diagnostic Session. It does that by sending a DevMgtSet(DiagSession) request to the DM that indicates both the scope and severity of the diagnostics to be performed. The manager replies specifying the Diagnostic Token that the diagnostic application will use to perform the diagnostics. However, this response does not give the diagnostic application permission to perform the tests.

The DM informs all of the affected client platforms and then sets up the Diagnostic Session with the IOU by sending it a DevMgtGet(DiagSession) specifying the Diagnostic Token for the session as well as the bounds (scope and severity) for the session. The DevMgt Agent indicates in the DevMgtGetResp(DiagSession) whether it is ready to proceed. If it is not ready, it will subsequently send a DiagSessionState trap when it is ready. Once the IOU notifies the DM that it has prepared for the diagnostic testing, the DM informs the diagnostic application that it may proceed via sending a Report(DiagSessionState) to the diagnostic application that requested the session. The diagnostic application may now perform diagnostic tests by suppling the Diagnostic Token in the MADs that it sends to the IOU, to validate that it is the one authorized to perform the tests. The IOU rejects any test that is not performed under a valid Diagnostic Token or any test that exceeds the bounds established for that Diagnostic Session.

In general, device diagnostics should be used with great care, and with full understanding of the potential impact to I/O transactions to the target device. It is best used during periods of initial configuration, major maintenance, or as a tool of last resort.

When the diagnostic application has completed its testing, it cancels the session by sending a DevMgtSet(DiagSession) to the DM. The manager invalidates the session by sending a DevMgtSet(DiagSession) to the IOU and then notifying the affected clients that the IOCs are back on-line. If the IOU responds with a DevMgtGetResp(DiagSession) with DiagStatus = not ready, then the DA does not notify the affected clients until the IOU sends a DiagSessionState trap indicating the session has terminated.

If the diagnostic application fails for any reason, the diagnostic lease expires and the manager performs the same actions as if the application had terminated the session. Likewise, if the manager fails for any reason, the DevMgt Agent automatically terminates the session when the lease expires and returns the affected IOCs to service.

The diagnostic framework allows for diagnostics to be performed at various levels (i.e., TestTargetType = a Service Object, an IOC, an I/O module, or the entire IOU). When a DevMgt Agent receives a DevMgtGet(DiagCode), DevMgtGet(DiagnosticTimeout), DevMgtSet(TestDevcieOnce), or DevMgtSet(TestDeviceLoop) specifying a TestTargetType that is not supported, the DevMgt Agent rejects the request using the MADHeader:Status of 'InvalidDiagTest'.

A8.5.1 BEHAVIORS

The DevMgt class of MADs (see A8.3 Device Mgt MAD Specification on page 1530) is used for diagnostics, i.e., DevMgtGet(), DevMgtSet(), and DevMgtTrap(). Vendor specific testing may be accomplished via the standard DevMgt methods and attributes or via vendor defined methods and/or attributes. In addition, standard DevMgt diagnostic attributes provide for additional vendor-specific data.

Each diagnostic attribute indicates the device under test. The DiagSession attribute can specify the IOU, a list of I/O modules, a list of IOCs, or a list of Service Objects - while the TestDeviceOnce, TestDeviceLoop, DiagnosticTimeout, and DiagCode specify either the IOU, an I/O module, a single IOC, or a single Service Object.

A diagnostic application may send DevMgtGet(DiagnosticTimeout) prior to establishing a Diagnostic Session to help determine the lease period. The lease period granted may differ from that requested so the application should check the GetResp() to determine the actual lease period. If more time is required, the diagnostic application may renew the lease periodically before the lease expires by sending another DevMgtSet(DiagSession) with Action = Renew.

Once the diagnostic application establishes a Diagnostic Session, it performs tests by sending DevMgtSet(TestDeviceOnce) or DevMgtSet(TestDeviceLoop) directly to the IOU.

Results are reported in the DiagCode attribute. The first 16-bits provide an overall status of the device under test where a value of zero indicates that the device is operational and all other values are vendor-specific. The Vendor-Specific component of the attribute data provide for additional vendor-specific information. The format of the Vendor-Specific component is defined by the IOU vendor.

A diagnostic application written by the vendor is able to perform any number of tests using vendor-specific fields. However, a diagnostic application not familiar with the product can still perform ‘default’ diagnostic testing. Default diagnostic tests are defined by the vendor as a means to perform a Go/NoGo test.

A8.5.2 PREPARING FOR DIAGNOSTIC TESTS

The DevMgtSet(DiagSession) attribute specifies the scope of the test by listing the service objects, IOCs, or I/O modules that may be tested. It also specifies the intensity/severity of testing that is allowed to be performed. The IOU places the device(s) into a test-ready state. The time required to complete this step is device-specific, as it may involve flushing data from cache memory, re-initializing SCSI ports, etc. If the IOU responds with a status indicating more time is required, the IOU subsequently indicates its readiness for test by having the DevMgt agent send a DiagSessionState Trap.

Alternatively, a DevMgtGet(DiagSession) also returns information pertaining to the specific device's ability or readiness for test. This allows the status to be polled in case the trap is lost.

CA8-95: When the DevMgt agent receives a DevMgtSet(DiagSession), it shall respond immediately[53] with the appropriate DiagSessionStatus as specified in Table 485 DiagSession on page 1598. If the preparation requires additional time, the DevMgt agent shall report “Device not ready” DiagSessionStatus.

CA8-96: It the DevMgt agent responds to a DevMgtSet(DiagSession) with DiagSessionStatus = “Device not ready”, then, when the status changes, the DevMgt agent shall generate a Trap as per Table 468 Notice 0x0801 DataDetails [DiagSessionState] on page 1556.

A8.5.3 INVOKING DIAGNOSTIC TESTS

After the diagnostic application successfully establishes a Diagnostic Session, it sends DevMgt MADs directly to the IOU’s DevMgt Agent to perform the diagnostics and get their status. The diagnostic application supplies the DiagToken in the MAD’s Access_Key field.

Two modes are provided for initiating diagnostics: single test mode and continuous test mode. In single test mode, a single test sequence is initiated by sending a DevMgtSet(TestDeviceOnce) attribute. The attribute specifies the target device and specific test. Which components of an IOC are testable is implementation-specific.

The DevMgt Agent rejects a DevMgtSet(TestDeviceOnce) that violates the bounds established by the Diagnostic Session. Once initiated, this vendor-defined test will run to completion. Because tests will vary by device technology and by vendor, the time-to-completion is inherently device-specific. To detect devices which are unable to complete their diagnostic test, a DiagnosticTimeout attribute may be retrieved in advance of test initiation, which indicates the maximum allowable period for

53. Within the response time indicated in ClassPortInfo.

completion. Results of the completed diagnostic test are obtained through the DevMgtGet(DiagCode) method, - i.e., diagnostic status is returned in the DevMgtGetResp().

The continuous test mode can assist in detecting problems that are transient in nature, or used to initiate endurance-related tests. The continuous-test mode is initiated by sending a DevMgtSet(TestDeviceLoop). Results of the last completed diagnostic test are obtained through the DevMgtGet(DiagCode) method.

For graceful termination of the continuous test mode, the initiator sends a DevMgtSet(TestDeviceLoop) with Action = 'Halt after current pass'. In this case, the DevMgt Agent terminates the continuous test mode after it completes the current pass and the DiagCode is valid.

The DevMgt Agent aborts the continuous test mode when it receives a DevMgtSet(TestDeviceLoop) with Action = Abort. In this case the DiagCode is indeterminate (e.g., might be set from the previous pass, might have been set to an initial value, or might indicate an intermediate result).

A8.6 IOC GRACEFUL HOT REMOVAL

"Surprise Hot Removal" is defined as the removal of an I/O module from an IOU without the module first being placed in a quiescent state.

"Graceful Hot Removal" is the removal of an I/O module that has first been placed in a quiescent state where no activity is present that would cause disruption upon module's removal. Module power might or might not be on. Other modules remain operational. The features required for the achievement of the quiescent state are described in this section.

Support for Graceful Hot Removal is optional. To claim compliance, an IOU must meet the requirements specified in this section and provide a DevMgt agent that supports Graceful Hot Removal.

This section covers removal of an I/O module (and its IOCs). An I/O module is different from an IB module because the I/O module does not contain the channel adapter. That is, the I/O module resides behind the channel adapter and thus DevMgt is still functional after the I/O module has been removed. Removal of IB modules (i.e., the IOU itself) is covered in Volume 2 under Baseboard Management.

Graceful Hot Removal requires that the I/O module has first been placed in a quiescent state where no activity is present that would cause disruption upon its detachment. The quiescent state is under control of the DM (i.e., a node that knows the IOU's Manager_Key). The DM's role is to coordinate the removal of an I/O module with clients that might be using to the affected Service Objects.

The term “affected clients” refers to those elements that are dependent on the presence of a given I/O module (i.e., its IOCs and their Service Objects) for operation.

The architecture assumes that there is a switch or lever on each I/O module that an operator engages to signal that the module is to be removed. That action causes a STATUS LED on the module to blink and a trap to be sent to the DM, who informs affected clients.

The request to remove may also be initiated via the DM allowing a software process such as a console application invoked by the system administrator, a diagnostic or health monitoring application, to inform the DM that an I/O module needs to be removed. In this case the DM sets a bit in the SlotControlStatus attribute indicating the module may be removed, which also causes its STATUS LED to blink.

When the affected clients have notified the DM that they have acquiesced their I/O operation, the DM sends an approval to the IOU, which causes the STATUS LED to turn off, indicating that it is OK to remove the module.

Power to the module is assumed to be removed when the LED is turned off.

A8.6.1 REQUIRED HOT PLUG FACILITIES

The following facilities and signals are used for interaction with the DM to accomplish Graceful Hot Removal:

- **Indicator LEDs** - Each I/O module has a Status LED and Attention LED as per A8.6.4 I/O Module Indicators on page 1622**.**
- **SlotControlStatus Attribute** - The DM uses this attribute to get and set information about an I/O module’s removal state. Each I/O module has the following set of bits:
  - **IOU_RTR** (IOU Request to Remove) - this is an indication that someone at the IOU wants to remove the I/O module. How this signal is generated is implementation-specific. It can be cleared via a DevMgtSet(SlotControlStatus). The IOU initializes this bit to zero (cleared) at power-up or IOU reset (but not IOC reset).
  - **SW_RTR** (Software Request to Remove) - this is an indication that the DM is processing the removal of the I/O module. The removal might have been initiated by the IOU or by the DM. This signal is set and cleared via a DevMgtSet(SlotControlStatus). The IOU initializes this bit to zero (cleared) at power-up or IOU reset (but not IOC reset).

- **SW_CTR** (Software Clear to Remove) - this is an indication from the DM that the I/O module is in the acquiesced state and thus can be removed gracefully. This signal can be set (unclaimed/acquiesced) and cleared (claimed) via a DevMgtSet(SlotControlStatus.RemovalControl). The IOU initializes this bit to one (unclaimed) at power-up.

- The ***DevMgtTrap*** mechanism is used to notify the DM that a Request to Remove (IOU_RTR) is pending.

oA8-4: An IOU that implements IOC Graceful Hot Removal shall support the SlotControlStatus attribute.

oA8-5: An IOU that implements IOC Graceful Hot Removal shall support the IomRemoval TRAP (Table 467 Notice 0x0020 DataDetails [IomRemoval] on page 1555).

A8.6.2 OPERATION

To enable Graceful Hot Removal, it is necessary for software to “Claim” the I/O module by clearing the **SW_CTR** bit, which turns the STATUS LED on indicating that software is presently using or preparing to use one or more of the module’s IOC.

At power-up, **SW_CTR** is set to 1b. Once the **SW_CTR** bit is set to 0b, the I/O module’s STATUS LED indicates that it is not OK to Remove the I/O module (See A8.6.4 I/O Module Indicators on page 1622 for a full description).

Once an I/O module is claimed, the general steps of Graceful Hot Removal are:

1) 1) The IOU or DM initiates a request for removal.

   a) For the case that the DM initiates the removal, the DM issues a ***DevMgtSet(SlotControlStatus:RemovalControl* = 0x02 = Set SW_RTR to one)** for the affected I/O module prior to collecting the necessary responses from the affected clients. Setting SW_RTR=1 instructs the DevMgt agent to indicate the “transition” condition through the blinking of the module’s Status LED (see A8.6.4 I/O Module Indicators on page 1622).

   b) For the case that the request is coming from the IOU, the DevMgt agent sets **IOU_RTR** for the affected I/O module and then forms and sends a **DevMgtTrap** out each port (if the port’s ClassPortInfo trap information has been set). On setting the IOU_RTR, the DevMgt agent blinks the module’s Status LED.

      Upon receipt of a **DevMgtTrap** or equivalent polling means that indicates **IOU_RTR** is set, the DM issues a **DevMgtSet(*SlotControlStatus:RemovalControl* = 0x02 = Set SW_RTR to one)**.The

setting of the **SW_RTR** bit to 1b acts as an acknowledgement to the setting of **IOU_RTR**.

2) The DM notifies all affected clients of the removal request and determines when all such entities have released their dependence on the IOCs. This is termed “Claim Release”.

3) For the case that not all affected clients concur with the claim release, the DM **shall** issue a **DevMgtSet*(SlotControlStatus:RemovalControl* = 0x01 = Reset IOU_RTR and SW_RTR to zero*)*** setting SW_RTR=0 to cancel the request.

   For the case that not all responses are received within a DM specific time-out period, the DM **may** issue a **DevMgtSet*(SlotControlStatus:RemovalControl = 0x01)*** to cancel the request. Failure to issue this operation will leave the IOC in a transition state for an indefinite period of time; the LED state for this case indicates that the I/O module is not OK to Remove.

4) For the case that all affected clients have quashed their operation and released the IOCs, the DM issues a **DevMgtSet*(SlotControlStatus:RemovalControl* = 0x04 = Set SW_CTR to one and reset IOU_RTR and SW_RTR to zero*)***. Setting SW_CTR=1 results in the I/O module Status LED condition being changed to the “OK to Remove” indication.

5) Anytime an I/O module is removed or added, the IOU sends a Slot Status Change trap to the DM.

Power is assumed to be removed when SW_CTR transitions from 0 to 1. How power is removed is implementation specific. Note that SW_CTR=1 is the default state at power up, but power is expected to be applied to the I/O module. In the case where there is no DM, the I/O module is never claimed and thus the Status LED remains OFF, indicating that the module may be removed (since there is no DM to control the removal sequence).

Graceful or ungraceful removal of an I/O module results in its IOU_RTR signal being reset.

A8.6.3 STATE DIAGRAM

The Graceful Hot Removal process is summarized in the state diagram of Figure 321: Graceful Removal State Diagram and generally depicts the behavior of the I/O module’s Status LED. (See A8.6.4 I/O Module Indicators on page 1622).

.

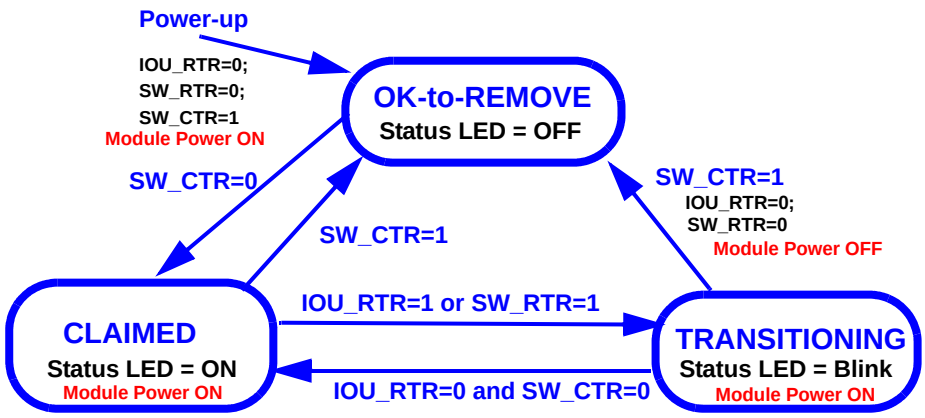


Figure 321 Graceful Removal State Diagram

A8.6.4 I/O MODULE INDICATORS

An IOU that implements IOC Graceful Hot Removal provides a Status LED for each I/O module to indicate the module's removal status.

oA8-6: An IOU that implements IOC Graceful Hot Removal shall have a Green Status LED and an Amber Attention LED for each removable module. The LED shall have the characteristics and operation as specified in A8.6.4 I/O Module Indicators on page 1622.

During power-on test, self-test, or diagnostics, the IOU/module may slow blink the Status LED as part of an LED test. Otherwise, LED states are controlled by the SlotControlStatus attribute bits IOU_RTR, SW_RTR, and SW_CTR as per Table 490: I/O Module Status LED

Table 490 I/O Module Status LED

IOU RTR	SW RTR	SW CTR	LED Condition	Description
x	x	1	OFF	I/O module can be gracefully removed
n/a	n/a	n/a	Slow Blink	LED test
1 X	X 1	0 0	Blink	Removal Transition - Release in progress but IOCs are still claimed and not removable.
0	0	0	ON	Claimed - I/O module can NOT be gracefully removed

A8.6.4.1 LED BLINK RATE DEFINITIONS

When used in this annex, the LED Conditions defined in Table 490: I/O Module Status LED **shall** apply to the repetitive ON/OFF behavior of the LED indicators as per Table 491: Blink Rate Definitions.

Table 491 Blink Rate Definitions

Term	Frequency	Period	Duty Cycle	Comments
OFF	0	n/a	**0%**	LED is steady off
Slow Blink	1/4 Hz	4 sec	**50 +/- 2 %**	Nominally 2 sec ON, 2 sec OFF
Blink	2 Hz	500 msec	**50 +/- 2 %**	Nominally 250ms ON, 250ms OFF
ON	0	n/a	**100%**	LED is steady on

A8.6.4.2 LED COLOR

oA8-7: The I/O module Status LED shall be Green or Blue-Green (555 - 565 nm).

oA8-8: The I/O module Attention LED shall be Amber (582 - 592 nm).

A8.7 DEVICE MANAGER

The role of the Device Manager is specified in the Configuration Management Annex. Device Management plays an intricate role in configuration management. A client platform communicates with the Configuration Management Application using DevAdm class to learn of IOUs and then uses Device Management to communicate with the IOUs. The Configuration Management Application uses Device Management to communicate with I/O units and configure them with client information. Additionally, third parties (such as other managers and diagnostic programs use Device Management to communicate with the DM to request a diagnostic session.

The DevMgt class provides mechanisms (methods and attributes) to enable a DM to set and retrieve DevMgt information from IOUs that provide a DevMgt agent. Other interested parties are also allowed to get DevMgt information directly from the IOUs. A DM configures an IOU using attributes listed Table 453 DevMgt Agent Attribute / Method Map on page 1542.

A DM communicates with the DevMgt agent by sending a DevMgtSet() or a DevMgtGet() to the DevMgt agent and the DevMgt agent responds with a DevMgtGetResp(). The DevMgt agent also sends DevMgtTrap()s to the DM and the DM responds with a DevMgtTrapRepress().

A8.8 I/O UNIT IMPLEMENTATION

DevMgt supports various concepts such as virtualized I/O resources. An IOU is modeled as one or more IOCs with each IOC providing one or more I/O services (i.e., I/O service object) via an I/O protocol. An IOC can support multiple I/O protocols. In addition to I/O protocols, the IOU or each IOC might also support one or more I/O management protocols. Thus, the term *Service Object* applies to both I/O management objects and I/O service objects.

I/O Service Object is an abstract term that refers to a service object with which an I/O client communicates to perform I/O operations. thus, the client uses a different QP for each service object. The architecture places no restriction on how an IOC defines service objects. For instance, an IOC might have:

- just one service object;
- a service object for each physical port to a secondary fabric;
- a service object for each virtual controller;
- a service object for each physical I/O device.

The following example illustrate the flexibility of Device Management by illustrating different ways a SCSI storage controller's Device Management information could be implemented.

For the following examples, assume an IB/SCSI device that has a single SCSI port, which connects to a SCSI bus with 3 SCSI devices. Device A and Device B have one logical unit (LUN 0) and Device C has 3 logical units (LUN 0,1,2).

Example 1: Talk through model where each SCSI device is a service object (i.e., each SCSI device appears as a SCSI target port). In this case, there would be three I/O service objects, each corresponding to one of the SCSI devices. A client assigned to the service object for Device C would have access to all three of its LUNs unless an I/O management application configured the object for LUN masking (i.e., managing which clients may access each LUN). The client would use a different QP to access each SCSI device.

Example 2: Talk-To model where the SCSI port is the I/O service object (i.e., a single SCSI target port). For this example, each I/O device (SCSI device) is presented as a protocol object and is ac-

cessed as a SCSI LUN, such that the I/O client sends SCSI commands to the I/O service object, and the LUN in the SCSI command block determines which I/O device is accessed. The 5 logical units could be represented by hierarchical LUN addressing (i.e. LUN 1.0, 2.0, 3.0, 3.1, 3.2). A client authorized for the service object would have access to all the LUNs, unless an I/O management application configured the object for LUN masking. The client sends all SCSI commands to the same QP.

Example 3: Virtual target devices. Each of the LUNs in example 2 are represented as a service object. For this case LUN masking is inherent in the virtualization. The client would need a different QP to access each LUN.

Example 4: Virtual target ports. For each client, I/O management creates a service object and the specifies which devices and LUNs may be accessed via that service object. For this case LUN masking is the virtualization. The client sends all SCSI commands to the same QP.

When the mapping between I/O devices and service objects is not fixed, or when the mapping varies based on the client, the mapping function (such as LUN masking) is typically provided by an I/O management protocol (via an I/O management object or ClientPoolRecord).

A8.9 COMPLIANCE

A8.9.1 COMPLIANCE CATEGORIES

In order to claim compliance to the Device Management class an I/O unit shall meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support.

Table 492 Device Management Compliance Qualifiers

Qualifier	Description
V1:	Backward compatibility with class version 1
HOT:	Hot plug / hot removal

A8.9.2 DEVICE MANAGEMENT AGENT COMPLIANCE SUMMARY

In order to claim compliance to the InfiniBand Specification for the Compliance Category of Managed IOU, a product shall meet all requirements specified in this section and in A8.9.3 “Common Management Requirements” on page 1628, except for those statements preceded by Qualifiers that the product does not support.

- **CA8-1:** Datagrams conform to DevMgt format . . . . . Page 1530
- **CA8-2:** Datagrams Follow Common MAD Use . . . . . Page 1531
- **CA8-3:** Response to the MADHeader:ClassVersion . . . . . Page 1532

A8.9.3 COMMON MANAGEMENT REQUIREMENTS

In addition, a Device Management Agent must also be compliant with the Common MAD requirements specified is 20.14 and the following general management framework requirements from Chapter 13.

ANNEX A9: VERB EXTENSIONS ANNEX

A9.1 INTRODUCTION

A9.1.1 OVERVIEW

This Annex defines enhancements to the Verbs specified in Release 1.1 of the base specification:

- efficient memory registration mechanisms for privileged Consumers,
- a mechanism for invalidation of Memory Windows and registered Physical Memory Regions,
- finer Memory Window protection through the ability to associate a bound Memory Window to a single QP,
- efficient mechanism for posting multiple work requests at one time,
- ability to share Receive Queue resources between multiple Reliable Connected and Unreliable Datagram Consumers through the use of a Shared Receive Queue (SRQ),
- ability to use multiple completion event handlers per HCA,
- ability to directly use host physical addresses through a Reserved L_Key, and
- introduction of zero based virtual addressing and block list oriented physical buffer lists.

The changes included in this Annex have been incorporated into the main body of this specification. Compliance statements for the Verb Extensions listed above take two forms:

CX-y.2.z: This type of compliance statement is mandatory in order to comply with the 1.1 specification semantics.

oX-y.2.z: This type of compliance statement is required to comply with the optional function referenced by the compliance statement.

Where X is dependent on which chapter and y & z or dependent on where the compliance statement is placed.

ANNEX A10: CONGESTION CONTROL

A10.1 CONGESTION CONTROL IN INFINIBAND NETWORKS

A10.1.1 GLOSSARY

Backward Explicit Congestion Notification	A signal, set by a destination port in either an ACK or a CNP. The BECN bit tells the source port that a packet it sent earlier, which corresponds to this ACK or CNP, encountered congestion.
BECN	Backward Explicit Congestion Notification
BTH	Base Transport Header
CA	Channel Adapter.
CCT	Congestion Control Table
CCTI	Congestion Control Table Index
CCTI_Increase	The amount by which a CCTI will be increased, when a packet which was marked as having gone through a point of congestion, is received.
CCTI_Limit	CCTI_Limit is the bounding value for a CCTI; CCTI cannot be greater than CCTI_Limit.
CCTI_Min	This is the lowest value that CCTI can be reduced to; the default value is zero.
CCTI_Timer	Congestion Control Table Timer
CCMgr	Congestion Control Manager see Congestion Control Manager
CCMgtA	Congestion Control Management Agent see Congestion Control Management Agent
Channel Adapter Congestion Event	The arrival of a packet with the BECN bit active in the BTH.
Channel Adapter Threshold Congestion Event	When a CCTI is equal to the Trigger Threshold this event is triggered and information pertaining to the event is logged by the CA.
CN	Congestion Notification
CNP	Congestion Notification Packet

Congestion Control Table	The Congestion Control Table contains an array of values of injection rate delay used to control congestion. The data is organized such that the lowest IRD is contained in entry 0, and the highest IRD is contained in the last entry of the table.
Congestion Control Table Index	An Index into the CCT, to select an Injection Rate Delay value.
Congestion Control Table Timer	A cyclic timer set up by the congestion manager, associated with an SL, which on expiry is reset and decreases the CCTI.
Congestion Control Manager	A manager dedicated to the control of congestion within a subnet. There may be more than one per subnet.
Congestion Control Management Agent	A management agent that decodes Congestion Control MAD attributes and applies them to a CA or Switch.
Congestion Notification	A Congestion Notification is generated to signal a congestion event to a source.
Credit Starvation	A mechanism to control the injection rate of legacy devices (e.g. CAs) by starving the device of credits.
CS_ReturnDelay	Used by Credit Starvation to control the rate of credit return upstream, while the input port is considered to be in a congested state.
CS_Threshold	The level at an input port, which when crossed, places the port in a congested state.
FECN	Forward Explicit Congestion Notification
Flow	A flow is a stream of data, from a QP or an SL (on a port).
Forward Explicit Congestion Notification	A signal set in a packet by a switch, to indicate that the packet encountered congestion along the path to the destination.
Injection Rate Delay	A minimum delay which a CA is to place between packets, to control congestion within a subnet. (See also CCT[CCTI]).
IRD	Injection Rate Delay
MAD	Management Datagram
QP	Queue Pair.
Root of Congestion	A switch output port whose buffer threshold for a VL has been exceeded, and for which there are link flow control credits available to send data continuously, is the root of congestion.

SL	Service Level
Switch Port VL Congestion State	The state of a VL at a port, in which packets for that VL are candidates for being marked with a FECN.
Switch Congestion event	When a VL on a switch port moves into a Switch Port VL Congestion State.
Trigger Threshold	When the CCTI reaches this level for a given flow, a Channel Adapter Threshold Congestion Event occurs.
Victim of Congestion	A switch output port whose buffer threshold for a VL has been exceeded, and which does not have sufficient flow control credits available to keep the data flowing at full rate, is a victim of congestion.
VL	Virtual Lane.

A10.1.2 CONGESTION OVERVIEW

To understand the congestion spreading problem, and the relationship between a root source and a victim of congestion, consider Figure 322 Fabric Congestion on page 1633. Assume that ports A through E on Switch 1 are all sending packets to port G so that port G is receiving data at 100% of its capability to send it.

Also assume that port F, on adjacent Switch 2 is also transmitting data to port G on Switch 1 at 20% of the total link bandwidth. Since egress port G is backed up, port F will transmit packets until there are no more credits left on the inter-switch link between the two switches.

At this point, port G will be congested, however there is no detrimental side effect, since all ports (A-F) will be served as quickly as port G is able to.

Now consider a port X on Switch 2, which is sending packets to port Y on Switch 1, at 20% of the paths bandwidth. Port G, the source of the congestion is not anywhere in the path from port X to port Y. In this case one might expect that since port F was only using 20% of the inter-switch links bandwidth, the remaining 80% of the links bandwidth would be available for port X, much more than port X requires. However this is not the case, since port F will eventually consume all of the link credits to switch 2, and port X will have to wait for available credits.

Congestion Control further defines two subcategories of congestion a port may experience. It may be the “root” cause of the congestion, or it may be a “victim”.

If a port has a large queue of packets awaiting transmission, and the port has available credits to send packets, then it is the root cause; however if the port does not have credits to send packets, then it is a “victim” of congestion spreading.

In this example port G in switch 1 would be the “root” of the congestion. It has available credits, however the queue is growing because it is receiving packets faster than it is able to forward them.

The congestion in Switch 1 has spread so that port H on Switch 2 is now congested. Port H will not have flow control credits, causing it to throttle data and is therefore a “victim” of congestion spreading. Traffic from port X is suffering from head-of-line blocking, and results in a reduction of the total throughput of the fabric.

The objective of congestion Control is to avoid if possible, and eliminate if it has already occurred, congestion spreading such as this. The approach here is to limit the injection rate of flows at the ports which are the root cause of the congestion (ports A-F), so that other ports are not affected (port X).

By limiting the injection rate of ports A-F to something which port G can handle, ports A-F should not see a significant degradation (after all, their packets were just going to wait anyway), however packets being sent from port X to port Y should be able to flow normally, since credits will be available.

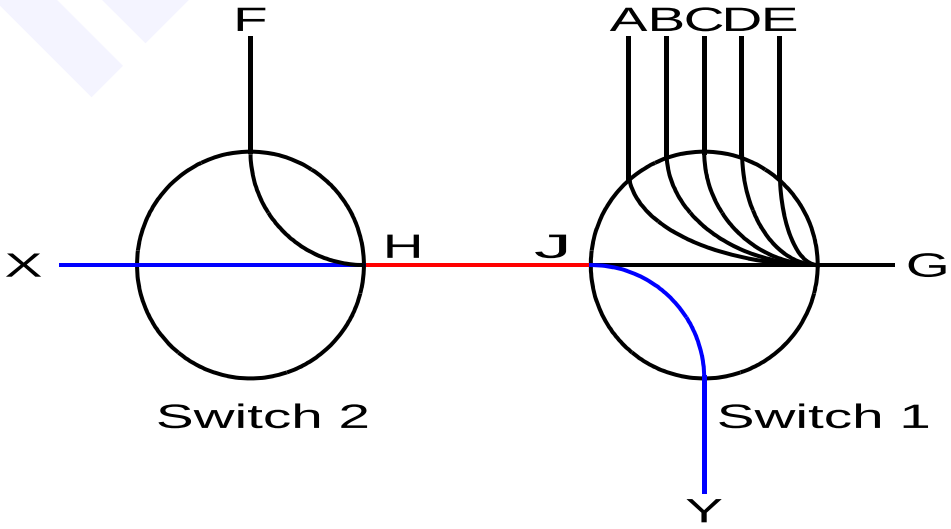


Figure 322 Fabric Congestion

There are situations where congestion can occur, yet no switch will detect that it is the root of congestion. For example, consider the case of a switch port connected to a CA that is unable to process received packets at the full line rate. In this case the switch output port buffer

threshold for a VL will be exceeded, and it will not have sufficient flow control credits available to keep the data flowing at full rate. As a result, the switch will view itself as the victim of congestion, instead of the root.

To handle this situation, a switch may be configured to enter the congestion state even when it views itself as the victim.

A10.1.3 CONGESTION CONTROL SUMMARY

This is a summary and is included to provide an overview of the congestion control specification. The normative portion of this specification is contained in A10.2 Congestion Control Mechanism on page 1636. A Congestion Manager is not architected by this specification.

- Switches detect congestion on a VL.
  - Switches distinguish between when that congestion is the root cause, or the result of congestion spreading (victim).
- Forward Explicit Congestion Notification (FECN) is used to communicate that a switch port is in the congested state.
  - Ports on switches may be configured to be in a congested state either when the port is the root of the congestion, or any time congestion is detected (i.e. the port is either the root or victim).
- The switch will set a FECN bit on a subset of the packets exiting the port when in the port VL congested state.
- The target CA of the marked packet then notifies the packet's source that congestion has occurred via a BECN.
- The source of the congested packet reacts by temporarily reducing its injection of packets into the network.
  - The original injection rate resumes over time.
  - Congestion Control may be performed on a per QP basis.
  - Congestion may alternatively be controlled for each SL on a Port.
- Cyclic logs are kept in congestion control aware switches and CAs, and are supplied to a congestion manager on request.
- Congestion control as defined will handle transient congestion within the fabric directly, and will define attributes, which may be used by higher levels of management to control longer term congestion, possibly in conjunction with a QoS manager.
- Security: Access to congestion control information in a switch or CA will be done under a congestion control key. See A10.4.1.1 CC_Key on page 1650 for detail of the CC_Key.

- For a link controlled by static rate control, if there is no congestion, static rate control will be predominant. When congestion occurs on a path that is controlled by static rate control, then the minimum of the two rates (i.e. the maximum interpacket delay) is used to inject packets into the fabric.

A10.1.3.1 CURRENT PERFORMANCE METRICS

Performance counters pertaining to congestion will be kept by CAs and switches.

Information that is currently collected may be pertinent to congestion:-

Mandatory counters:-

- PortXmitData. The closer this is to the maximum transmission of a link the more likely it is that a port may be in a congested environment.
- PortXmitWait. This provides information on a potential victim of congestion. It is the number of ticks during which the port had data to transmit but was unable to because of a lack of credits or a lack of arbitration.

Optional counters:-

- PortXmitQueue[n]
- PortXmitDataVL[n]
- PortVLXmitWaitCounters:PortVLXmitWait[n]
- SwPortVLCongestion:SWPortVLCongestion[n]

For Further information see 16.1 Performance Management on page 930.

A10.1.3.2 OPERATION WITH REV 1.1 SWITCHES AND CHANNEL ADAPTERS

The full capability of the congestion control mechanism requires congestion control aware CAs and switches. However, a mix of congestion control aware and unaware devices is supported, with, at most, some loss of Congestion control.

Marked packets and Congestion Notification Packets (CNPs) traverse release 1.1 (or earlier) switches as valid packets.

The Opcode in the BTH of a CNP has been chosen so that Release 1.1 (or earlier) CAs ignore the congestion notifications. The FECN and BECN are in fields that are reserved in the Release 1.1 specification.

A10.2 CONGESTION CONTROL MECHANISM

The implementation of congestion control in a Switch or a CA is optional. When implemented, it shall be done so in accordance with the specification in this section.

In order for congestion control to work effectively with unreliable datagrams, the paths must be reversible as described in 13.5.4 Response Generation and Reversible Paths on page 768. CN's can only be constructed from the received datagram with a FECN mark (see A10.3.2 on page 1648). BECNs may not reach the source if the path is non reversible, as they may be dropped.

Congestion control is only applicable to Data VLs; it does not apply to VL15.

A Congestion Control Manager will identify that a switch or CA supports Congestion Control by receiving successful responses to valid Congestion Control Method requests. A non congestion aware CA or switch will return either "Bad Version" or drop the packet. If a request times out it is up to a congestion manager to determine how many times it will retry without a response before the CA or Switch is assumed not to support Congestion Control.

CA10-1: A CA or Switch that supports Congestion Control shall so-indicate by responding to valid Congestion Control Method requests successfully.

A10.2.1 SWITCH BEHAVIOR

This section defines the operation of a switch that supports congestion control.

A switch that supports congestion control may or may not support congestion control on its enhanced switch port zero. The indication of such support is by the setting of EnhancedPort0CC in the Congestion Control ClassPortinfo:CapabilityMask, see Table 501 Congestion Control ClassPortInfo:CapabilityMask on page 1655. The requirements are identical to those for a CA. Congestion control is not supported on base port0.

A10.2.1.1 CONGESTION DETECTION

A switch port may identify that it has entered a Port VL congested state when it crosses a threshold. The threshold for this event may be set to a different level for each port. The threshold is determined by a weight that has a value of 0 to 15 and is interpreted as follows:-

- 0: there is no marking of packets on this port

- 1: Indicates a high value of a threshold (little space to accommodate for packets already in the fabric).i.e. there is a high probability that congestion will spread.
- 2-14: A uniform distribution of thresholds between those assigned to weights 1 and 15.
- 15: indicates a low value of the threshold (more space to accommodate packets in the fabric). In this case the likelihood of congestion spreading is reduced, but there is an increased probability that a packet might be signalled as moving through a point of congestion, when the fabric is not congested.

The threshold should be considered in terms of credits. A threshold with weight 15 will be set by the device such that traffic with a fixed packet size of 256 bytes (including headers), with a uniformly random destination address on each supported VL on the port, and an inter packet gap of 176 bytes (approximate load of 60%), on all ports will have a <1% probability of packets being marked with a FECN. This calculation should be performed on a device configured such that each port has the same number of operational VLs.

This will provide different settings for the threshold with a weight 15, depending on the number of VLs that are configured on a port. Fewer VLs provides more buffer space.

A weight of 1 should have a threshold that has consumed the majority of the buffering for a VL

A switch should have a default weight of 8.

The weights from 15 to 1 should cover a uniform distribution over the available buffering from the average 60% queueing point, on a VL basis.

The setting of the thresholds in the switch is dependent on the architecture (Output buffered, Input buffered, central shared buffer pool with or without dynamic buffer allocation, virtual output queueing, etc.). The association of the congestion weighting to the threshold, beyond what is outlined in A10.2.1.1 Congestion Detection on page 1636, is left to the device designer.

CA10-2: A weight of 0 shall indicate no congestion marking.

CA10-3: A weight of n shall produce a threshold that is lower than a threshold generated by a weight of n-1.

CA10-4: There shall be 16 distinct thresholds levels within a device (based on credits), including the threshold for a weight of 0 which indicates no marking.

A10.2.1.1.1 ROOT VS. VICTIM

A switch is capable of identifying both root and victim of congestion.

- Identify a root of congestion:- An output VL has exceeded the selected threshold and there are credits available to output data, i.e. data is always able to exit, but may be delayed due to arbitration.
- Identify a victim of congestion:- An output VL has exceeded the selected threshold and packets are delayed due to a lack of credits, i.e. data may not always be able to exit the port due to a lack of credit.

Under certain circumstances, such as when a switch port is connected to a CA that is unable to process received packets quickly enough, it may be beneficial to enter the Port VL congested state when the switch identifies itself as a victim of congestion. This can be accomplished by setting the Victim_Mask for the port (see A10.4.3.6 SwitchCongestionSetting on page 1659). The Victim_Mask would typically be set for enhanced switch port zero.

The indication of congestion through the VL should be smoothed over time. The precise smoothing mechanism is left to the switch implementor. The precise mechanism for identifying congestion is dependent on the switch architecture.

CA10-5: A switch that supports congestion control shall be capable of determining whether it is a root or a victim of congestion.

CA10-6: A switch port data VL shall detect that it is a root of congestion when the congestion threshold has been exceeded and there is sufficient credit to transmit a packet at the head of the queue on that data VL. It is permissible to smooth or average credits in the determination of a root of congestion, as long as the result maintains the long term behavior.

CA10-7: A switch port VL shall detect that it is a victim of congestion when the congestion threshold has been exceeded and it is unable to transmit a packet at the head of the queue on that data VL due to a lack of credit.

A10.2.1.2 CONGESTION MARKING

A switch marks packets when a switch port is in a switch port VL congestion state which is defined as follows.

CA10-8: A switch port VL shall be in the congestion state when it is a root of congestion.

CA10-9: A switch port VL shall be in the congestion state when it is a victim of congestion and the Victim_Mask bit (see A10.4.3.6 SwitchCongestionSetting on page 1659) is set for that port.

CA10-10: A switch shall generate a switch congestion event when a port VL enters the congestion state. This event is used to log events in the switch CongestionLog (see Table 506 CongestionLog (switch) on page 1658).

CA10-11: A switch that has a port that enters a switch port VL congestion state, shall mark packets by setting the “F” field in the BTH, while in this state. See A10.3.1 BTH: FECN and BECN locations on page 1647

CA10-12: A switch shall not mark raw packets with a FECN.

CA10-13: A switch shall not mark packets with a FECN, if the packet size is below the specified value. If the packet_size threshold is set to 0, then packets of all size are eligible to be marked. To control the marking packet size see:- Table 510 SwitchCongestionSetting on page 1660 and Table 511 SwitchPortCongestionSetting Attribute on page 1661.

CA10-14: A switch shall set the FECN bit in the BTH of packets that are candidates for marking determined by the Marking_Rate. (see Table 510 SwitchCongestionSetting on page 1660 marking_rate). If the Marking_Rate is set to zero, then all packets are eligible to be marked.

A10.2.1.3 CONGESTION LOG

A switch that supports congestion control maintains a congestion log to identify the frequency and source of congestion events. Due to the possibility of contention in large switches this log is best effort, but while a switch is marking packets this log file shall have at least one active entry. See Table 506 CongestionLog (switch) on page 1658.

- Entries may be placed in the log for each Switch Congestion Event. This may happen on more than one VL on a port, and for more than one port in the same switch at the same time.
- Only a sampling of switch congestion events needs to be reported; all events are not reported.
  - If the switch marks multiple packets at any instant during a congestion period for the same SL/DLID, it may chose to record only one of the packets in it’s log of congestion events (best effort).
  - The switch maintains information for the last 20 congestion events. Some entries may be zero if either twenty events have not yet occurred, or if events are too old (outside the scope of the current timestamp).
  - The Switch Congestion Log is not cleared on reading, and is not explicitly cleared. A congestion manager identifies duplicate entries from successive reads via the time stamps.

A switch maintains a free running 32 bit timer for log purposes that encompasses a range of 1.024 µsec to 1.22 hours.

CA10-15: A switch that supports congestion control shall maintain a Switch Congestion Log.

CA10-16: The Switch Congestion log shall be a cyclic buffer of 20 entries.(i.e. most recent events replace older ones)

CA10-17: A switch shall place at least one entry in the switch congestion log when it generates one or more switch port VL congestion events.

CA10-18: A switch shall maintain a free running 32 bit timer for log purposes that increments in 1.024 µsec steps and is derived from the link bit rate.

CA10-19: The Switch Congestion Log shall contain the information defined in Table 506 CongestionLog (switch) on page 1658.

CA10-20: In a Switch Congestion Log, entries that are older than the maximum timestamp wrapping twice, shall be returned as zero.

A10.2.1.4 SWITCH PERFORMANCE COUNTERS FOR CONGESTION

The switch maintains the following sampled performance counters that pertain to congestion on a port.

- The length of time (number of ticks) during the sample period that the port for a given data VL is in the switch port VL congestion state (PortXmitTimeCong).
- The length of time (number of ticks) during the sample period that any data VL on a port is in the switch port VL congestion state (PortVLXmitTimeCong).

For a definition of tick, see 16.1.3.2 PortSamplesControl on page 933.

At 2.5Gbps a tick can be in the range of 4 nsec to 1 microsecond (in multiples of 4 nsec). If a higher resolution tick is used, then the counter shall be larger. For a 4ns resolution, the counter shall be 32 bits (max. of 4 seconds).

CA10-21: A switch that supports congestion control shall maintain the sampled performance counters to be returned to the CCMgr on request defined in Table 525 PortXmitConCtrl on page 1673 and Table 526 PortVLXmitTimeCong on page 1673.

A10.2.1.5 SWITCH CREDIT STARVATION

Switch Credit Starvation may be used to control congestion when attached to legacy devices (CAs or switches), or even “rogue” Congestion Control capable endpoints. It is an optional feature within Congestion Control.

The technique is to control the return of credits to the device upstream of a link (Credit Starvation), when an input port is marked as having such a mechanism active. The mechanism identifies a trigger point at which a port will start to control the rate of credit return for a VL. The trigger point will be based on an average “input buffer” threshold for a VL, but may also be considered as being the average active number of credits used for a VL. Note that this mechanism will affect flows that might not be going through the root of congestion. While this condition exists, the credit return rate will be controlled by the CS_ReturnDelay. When the average “buffer occupancy” for a VL falls below its threshold, credits will be returned by the switch’s normal policy.

There will be one average “buffer occupancy” threshold and a CS_ReturnDelay for each port. Each port on a switch may have different values. Capable ports can be set active, through the Credit_Mask, see Table 510 SwitchCongestionSetting on page 1660, which also contains the default settings for the trigger level (CS_Threshold), and credit return rate (CS_ReturnDelay) see Table 510 SwitchCongestionSetting on page 1660 for switch defaults, and Table 511 SwitchPortCongestionSetting Attribute on page 1661to set different rates on ports. Individual ports may be set, see A10.4.3.7 SwitchPortCongestionSetting on page 1661

oA10-1: A switch may support the Switch Credit Starvation option.

oA10-2: A switch shall indicate support of the Switch Credit Starvation option by setting Bit0 in the CongestionInfo to a one, see Table 504 CongestionInfo on page 1657

oA10-3: A switch that supports the Switch Credit Starvation option shall activate credit starvation on each port that has the Credit_Mask set to one (see Table 510 SwitchCongestionSetting on page 1660).

oA10-4: A switch port that has credit starvation activated shall detect when the CS_Threshold for a data VL on that port is exceeded.

oA10-5: When the CS_Threshold is exceeded on a switch port data VL, the credits for that VL shall be delayed as specified by the CS_ReturnDelay for that port.

A10.2.2 CA BEHAVIOR

The implementation of congestion control in a CA is optional. This section defines its operation if implemented.

A CA that supports congestion control is responsible for notifying the source when packets are received that indicate they have passed through a point of congestion. The CA is also responsible for reducing the injection rate of flows that pass through a point of congestion when it receives such a notification from the target.

A CA ignores the FECN bit in the BTH (it will not generate a BECN response) of the following packet types:-

- Multicast Packets
- ACK Packets
- CN Packets

The CA notifies the source of a packet that has passed through a point of congestion by sending a CN packet. Alternatively, for reliable connections, this may be optimized by placing a BECN marker in the ACK rather than generating a CNP. Note: ACKs are not normally subject to IRD, however an RDMA_Read_Response is an ACK which may be controlled by IRD, and in such a case, a CNP should be used to signal a BECN back to its source.

A CA may be subject to high rates of small packets arriving which are marked with a FECN and, additionally, there is the possibility of output scheduling delays. Under such conditions it may not be possible to generate a BECN for each FECN that arrives. Under such conditions a device shall:-

- For BECNs in ACKs, BECN responses may be coalesced.
- The BECN bit on a coalesced Acknowledge packet shall be active if one or more of the Acknowledges covered by the coalesced acknowledge should have had its BECN bit active.
- For BECNs in CN's, the endpoint shall be able to support at least 1 outstanding CN per QP, or if a centralized CN allocation mechanism is used, shall be capable of queueing at least 32 CN's per port. (When CN resources are exhausted a FECN shall have its corresponding BECN dropped).

On receipt of a FECN, the resulting ACK or CN should be scheduled as quickly as possible.

CA10-22: CAs that support congestion control shall ignore the FECN bit set in the BTH of multicast, ACK and Congestion Notification Packets.

CA10-23: When a valid FECN is received (see CA10-22:), the CA shall set the BECN bit to a one in the BTH of either an ACK or a CNP that is returned in response to the packet received with the FECN bit set.

CA10-24: A CA that supports congestion control shall be capable of scheduling at least one CNP per QP or 32 CNPs per port.

CA10-25: A CA that supports Congestion control shall be able to handle a BECN received in either an ACK (including unsolicited ACK), or a CNP.

CA10-26: A CA that supports congestion control and marks ACKs with a BECN shall schedule an ACK marked with a BECN regardless of the state of the AckReq bit in the received packet's BTH field.

CA10-27: CAs that support congestion control shall ensure that the BECN bit on a coalesced Acknowledge packet shall be active if one or more of the Acknowledges covered by the coalesced acknowledge should have had its BECN bit active.

A10.2.2.1 INJECTION RATE CONTROL

The following describes the way in which a congestion-control-capable CA receiving a BECN will control the rate of injection into the fabric.

A CA has one Congestion control table (CCT) per port, with a minimum of 128 entries in each CCT. The CCT may be larger, but only in multiples of 64 entries. A CA returns the size of the CCT in 64 entry units in ControlTableCap in the CongestionInfo attribute (see: Table 504 CongestionInfo on page 1657).

A10.2.2.1.1 CCT ENTRY FORMAT

A CCT entry is 16 bits wide; consisting of a 2-bit shift_field and a 14-bit multiplier. The lowest injection rate that can be achieved is 100/(2**14) or 0.006% of a links throughput.

A CA shall interpret a CCT entry as follows:-

The IRD will be the value of CCT[CCTI] that defines the delay between packets from this flow.

CA10-28: For a CA that supports congestion control, if a packet has been sent from a flow, the subsequent packet from the flow shall not be scheduled until *at least* time T_s has passed since the previous packet was scheduled, where T_s is calculated as follows:

Ts = (TpacketTime >> Shift_field) * Mult_Factor

Where:

>> = shift right

Multi_factor = CCT[CCTI] bits 0-13 (14 bit multiplier)

Shift_field = CCT[CCTI] bits 14-15

TpacketTime = time it takes to transmit the preceding packet

LRH:PktLen*4/$L_{r,}$

L_r is the port speed as obtained from the PortInfo:LinkWidthActive and PortInfo:LinkSpeedActive attributes.

A CA may aggregate a small number of packets together (up to 4 MTU) before asserting IRD. In such a case, the IRD applied between such grouping shall be that appropriate to the aggregate size.

A CA controls the injection rate of a flow on either a QP or SL basis, depending on the setting for the port. It shall either be QP or SL based exclusively. See Table 513 CACongestionSetting on page 1662

A CA will have a CCTI and CCTI_Min per flow (QP and SL). It also has a table for CCTI_Increase and CCTI_Timer for each port. Each table has 16 entries; one for each SL. See Table 513 CACongestionSetting on page 1662.

CA10-29: A CA that supports congestion control shall interwork with Static Rate Control. The IRD will be the maximum of the static rate control and the congestion rate control.

CA10-30: A CA that supports congestion control shall maintain one Congestion Control Table (CCT) per port.

CA10-31: The CCT shall contain a minimum of 128 entries.

CA10-32: A CA shall report the number of entries in the CCT in 64 entry units in the CongestionInfo attribute (see Table 504 CongestionInfo on page 1657).

CA10-33: A CA shall load the CCT based on the information received in the CongestionControlTable attribute (see Table 515 CongestionControlTable on page 1664).

CA10-34: A CA shall maintain a CCTI_Limit for each CCT.

CA10-35: A CA shall be capable of controlling the injection rate on each QP flow or on each port SL flow, depending on the setting of the Port Control:FlowSelect (bit 0) in CA CongestionSetting (see Table 513 CACongestionSetting on page 1662).

CA10-36: A CA shall maintain a CCTI for each QP or port SL flow.

CA10-37: A CA shall maintain a CCTI_Min, CCTI_Increase and CCTI_Timer for each port SL and shall load them based on the contents of the CA CongestionSetting attribute (see Table 513 CACongestionSetting on page 1662).

CA10-38: Each CCTI_Timer shall increment in 1.024 µsec steps and be derived from the link bit rate.

A10.2.2.1.2 RATE DECREASE

The injection rate is decreased for a flow (QP or port SL) based on the receipt of BECNs. When the CA is configured for SL based control, the CCTI is incremented by the CCTI_Increase value associated with the SL. When configured for QP based control, the CCTI_Increase value for the SL associated with that QP is used. The CCTI is only increased up to the CCTI_Limit, and no history of BECNs arriving while the CCTI is at the CCTI_Limit is kept.

CA10-39: When the CA is configured for SL-based control, the CCTI is incremented by the CCTI_Increase value associated with the SL, each time a BECN is received on that SL.

CA10-40: When the CA is configured for QP-based control, the CCTI is incremented by the CCTI_Increase value for the SL associated with that QP, each time a BECN is received on that QP.

CA10-41: A CA shall not increment the CCTI beyond the CCTI_Limit.

A10.2.2.1.3 RATE INCREASE

The injection rate is increased periodically based on the value set in the CCTI_Timer. The CCTI_Timer has a range from 1.024 µsec to 67 msec and is maintained for each port SL. When this timer expires, the CCTI for each flow associated with that timer is decremented by one, thus referencing a CCT entry that has a reduced delay value. When the CCTI reaches zero, no injection rate delay is to be added to the flow.

The CCTI is not reduced any further, once it has reached either CCTI_Min or zero (whichever is the larger).

This specification describes setting CCTI_Min for a port SL (see A10.4.3.8 CACongestionSetting on page 1662). When configured for QP-based injection rate control, the CCTI_Min for the associated SL is applied to the QP.

CA10-42: A CA shall decrement by one, the CCTI for each flow associated with an SL that has its CCTI_Timer expire. The CCTI shall not be decremented below zero or the CCTI_Min value for the flow.

A10.2.2.2 CA CONGESTION THRESHOLD EVENT NOTIFICATION LOG

A congestion-control-capable CA provides a Congestion Threshold Event Notification Log for each port in order to provide information pertaining to the frequency and source of congestion events. CAs provide the capability to detect when a CCTI reaches the Trigger_Threshold that is set by a Congestion Manager. When the Trigger_Threshold for a flow is reached, information pertaining to the event is logged by the CA. A Congestion Manager sets the Trigger_Threshold for each SL on a CA port using the CACongestionSetting attribute (see A10.4.3.8 CACongestion-Setting on page 1662). If the Trigger_Threshold is set to zero, no information is logged.

The Congestion Threshold Event Notification log is a cyclic buffer of sixteen entries. It is not cleared on reading, and is not explicitly cleared. A congestion manager identifies duplicate entries from successive reads via the entry time stamps.

A CA shall maintain a free running 32 bit timer for log entries, with a range of 1.024 µsec to 1.22 hours.

A CA places entries in the notification log as follows:-

- The detailed information recorded for each log entry is given in Table 509 CongestionLogEvent (CA) on page 1659. Event Log entries for which information is not available are returned as all zeros.
- The log maintains an indication that the threshold for any SL has occurred since the last CCMgtGet(CongestionLog). This indication for each SL is returned in the Trigger_Threshold Log information in the ThresholdCongestionEventMap field (see Table 508 CongestionLog (CA) on page 1658).
- When the ThresholdCongestionEventMap has been returned in a CCMgtGetResp(CongestionLog) it shall be set to zero. It is possible for a packet to get lost, and thus lose the current information.
- If multiple notification events occur in quick succession, it may not be possible to log all the events, in which case a CA will log as many as may be practical (best effort).
- When a CA is configured to control congestion on a port SL, the following entry fields shall be maintained:-
  - SL
  - Remote LID
  - Timestamp

CA10-43: A CA that supports congestion control shall provide a Congestion Threshold Event Notification Log for each port.

CA10-44: The Congestion Threshold Event Notification Log shall be a cyclic buffer of sixteen entries.

CA10-45: A CA shall detect when the CCTI for a flow is increased and becomes equal to the Trigger Threshold. When this is detected the CA shall generate a threshold event.

CA10-46: A CA shall place at least one entry in the Congestion Threshold Event Notification Log when it detects that one or more threshold events have occurred.

CA10-47: A CA shall maintain a free running 32-bit timer for log entries that increments in 1.024 µsec steps and is derived from the link bit rate.

CA10-48: The Congestion Threshold Event Notification Log shall contain the information defined in Table 508 CongestionLog (CA) on page 1658.

CA10-49: In a Congestion Threshold Event Notification Log, entries that are older than the maximum timestamp wrapping twice, shall be returned as zero.

A10.2.2.3 CA PERFORMANCE COUNTERS

A congestion-control-capable CA supports the following sampled performance counters on each port.

- The number of packets received with a FECN mark (PortPktRcvFECN).
- The number of CNPs/ACKs received with a BECN mark (PortPktRcvBECN).
- The number of packets for a specified SL received with a FECN mark (PortSLRcvFECN).
- The number of CNPs/ACKs for a specified SL received with a BECN mark (PortSLRcvBECN).

CA10-50: A CA that supports congestion control shall provide the sampled performance counters defined in A10.5.4 PortRcvConCtrl on page 1668, A10.5.5 PortSLRcvFECN on page 1669 and A10.5.6 PortSLRcvBECN on page 1671.

A10.3 PACKET FORMATS

A10.3.1 BTH: FECN AND BECN LOCATIONS

The following table shows the BTH and where the FECN and BECN will reside. The FECN & BECN bits are in the area of the BTH that is not covered by the iCRC.

Table 493 BTH Header

bits bytes	31-24			23-16				15-8	7-0
0-3	Opcode			SE	M	Pad	TVER	Partition Key	
4-7	F	B	Reserved-6	Destination QP					
	(masked in the iCRC)								
8-11	A	Reserved 7		PSN - Packet Sequence Number					

B (BECN): 0 indicates that no congestion was encountered, 1 indicates that the packet indicated by this header was subject to forward congestion. The B bit is set in an ACK or CN BTH.

F (FECN): 0 indicates that it probably did not go through a point of congestion, 1 indicates that the packet went through a point of congestion.

A10.3.2 CONGESTION NOTIFICATION PACKET (CNP) FORMAT

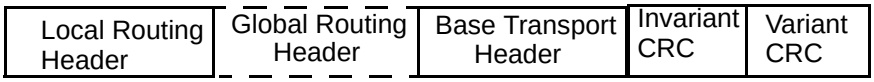


Figure 323 Congestion Notification packet format

Notes:

a) The Lver-field of the LRH must be set to 0x0. To be transparent to version 1.1 and earlier switches.

b) The BECN bit of the BETH will be set (see BTH bit settings above).

c) CNP opcode of 0b'10000000.

d) The QP field in the BTH will be set to the source QP of the packet marked with a FECN:-

   For UD the QP is obtained from the DETH.

   For UC and RC the QP is obtained from the QP context.

   For RD the QP is obtained from the EE context.

e) The PSN will be set to 0 and ignored by the receiver.

f) The P_Key will be the same as the BTH of the packet marked with a FECN.

g) The Tver in the BTH will be set to 0x0

h) The Solicited Event and Migration Required shall not be set.

i) If the original packet contained a GRH, then the CN packet must contain a GRH. For original packets being RC and UC this information is obtained from the QP context, for RD it is obtained from the EE context and for UD as follows:-

The DGID is copied from the SGID in the GRH of the original packet.

The FlowLabel and TrafficClass are copied from the original packet.

HopLimit is set to 0xFF

CA10-51: A congestion control aware device that signals or receives BECNs, shall support the CN packet format as described in section A10.3.2 Congestion Notification Packet (CNP) format on page 1648.

A10.4 CONGESTION CONTROL MANAGEMENT

CA10-52: The Congestion Control Management Agent (CCMgtA) is mandatory on all nodes that support congestion control.

The Congestion Control Management class provides mechanisms to configure congestion control parameters in switches and channel adapters that support congestion control. It also provides mechanisms for retrieving information pertaining to congestion events from such devices.

A10.4.1 CONGESTION CONTROL MAD FORMAT

CA10-53: The datagrams for the Congestion Control class shall conform to the MAD format and used as specified in 13.4 Management Datagrams on page 627 and further specified in Figure 324 on page 1649 and Table 494 Congestion Control MAD Fields on page 1650.

The management class for the Congestion Control class is 0x21.

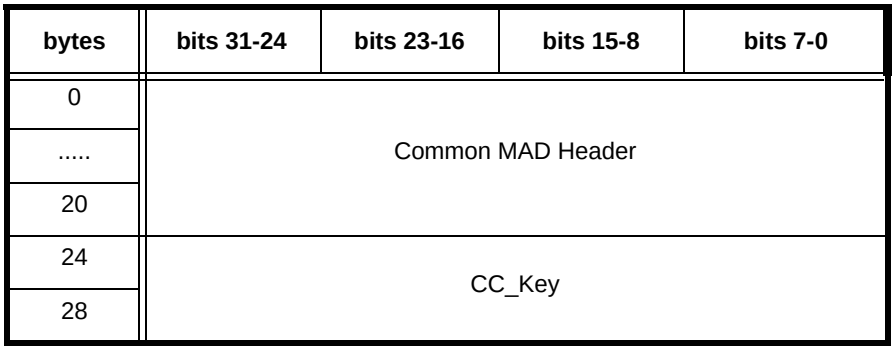


Figure 324 Congestion Control MAD format

bytes	bits 31-24	bits 23-16	bits 15-8	bits 7-0
32	Congestion Control Log Data (otherwise reserved)			
...				
60				
64	CCMgt Data			
...				
252				

Figure 324 Congestion Control MAD format

Table 494 Congestion Control MAD Fields

Field Name	Length	Description
Common MAD Header	24 bytes	Common MAD header as described in 13.4.2 Management Datagram Format on page 718. MgtClass0x21, ClassVersion=2.
CC_Key	8 bytes	Congestion Control key, is used to validate the source of the Congestion Control MADs, see A10.4.3.4 CongestionKeyInfo on page 1657 for definition and use.
Congestion Control Log Data	32 bytes	32 bytes of CCMgt payload for congestion control log MADs. This data may extend into the CCMgt Data field below. See A10.4.3.5 CongestionLog on page 1657 for attribute format.
CCMgt Data	192 bytes	192 bytes of CCMgt payload. The structure and content depends upon the Method, Attribute, and Attribute Modifier fields in the MAD header.

A10.4.1.1 CC_KEY

In order to authenticate that congestion control MADs originated from a trusted source, all Congestion Control MADs must include the Congestion Control Key (CC_Key).

A10.4.1.1.1 CC_KEY ASSUMPTIONS

6) To use the correct key for each node, a CCMgr or a higher level CC_Key manager keeps track of the keys for the nodes that it is managing.

7) If a backup CCMgr exists, it shares the CC_Keys for ease of fail-over.

8) A CCMgr sets the CC_Key, CC_KeyProtectBit, and CC_KeyLeasePeriod in the CCKeyInfo Attribute with one CCMgtSet(CCKeyInfo) MAD. A successful completion of this assignment indicates to a CCMgr that it has taken ownership of the node.

A10.4.1.1.2 CC_KEY PROTECTION SCOPE

Each Congestion Management Agent in a node has one CC_Key. All access from a CCMgr to a CCMgtA are protected by the CC_Key. Table 495

CC_Key Protection Scope on page 1651 shows the scope protected by the CC_Key.

Table 495 CC_Key Protection Scope

Source	Target Entity	Protection
CCMgr	Reads and Writes to a CCMgtA	yes

A10.4.1.1.3 CC_KEY OPERATION

CA10-54: The CCMgtA shall check the CC_Key contained in incoming MADs of the Congestion Control class.

The success and effect of the check depends on the value of the CCKeyInfo:CC_Key and CCKeyInfo:CC_KeyProtectBit of the CCMgtA and on the method and attribute contained in the incoming MAD.

Table 496 CC_Key Check

CCMgtA CC_Key	CCMgtA CC_Key Protection Bit	MADs method	Success
Zero	any	any	yes
non-zero	any	CCMgtSet()	if MADs CC_Key equals CCMgtAs CC_Key
non-zero	0	CCMgtGet()	yes
non-zero	1	CCMgtGet()	if MADs CC_Key equals CCMgtAs CC_Key[a]

a. Even though the check succeeds, the CC_Key value in the CCKeyInfo attribute shall be returned as zero

CA10-55: If the CC_Key check fails, the CCMgtA shall

- Drop the MAD.
- Increment a CC_Key Violation counter if supported.
- Send a CCKeyViolation trap or generate a CCKeyViolation Notice.
- Start a countdown timer with the CC_Key lease period value.

A10.4.1.1.4 CC_KEY INITIALIZATION

CA10-56: At power up or reset, the CCKeyInfo:CC_Key, CCKeyInfo:CC_KeyProtectBit and CCKeyInfo:CC_KeyLeasePeriod shall be set to zero.

A CCMgr may use CCMgtSet(CCKeyInfo) to assign the subsequent CCKeyInfo:CC_Key, CCKeyInfo:CC_KeyProtectBit and CCKeyInfo:CC_KeyLeasePeriod.

A10.4.1.1.5 CC_KEY RECOVERY

CA10-57: The CC_Key lease period timer shall start when a CC_Key check fails.

When a CC_Key check fails, the node either sends a trap to the CCMgr or generates a Notice indicating the attempted access. This trap or notice serves as a request to the CCMgr to refresh the lease period by issuing a CCMgtSet(CCKeyInfo). A successful CCMgtSet(CCKeyInfo) will stop the timer and will rearm it.

If the CCMgr that originally set the CC_Key has gone away, then the lease period expires --- clearing the CCKeyInfo:CC_KeyProtectBit and allowing a new CCMgr to read (and then set) the CCKeyInfo:CC_Key.

In the case where the TrapLID is zero (because no CCMgr has set it), the node has no CCMgr to send the trap to. In this case, the node does not send the trap and the lease period timer will expire, causing eventual take over by a new CCMgr

With the CCMgtGet(CCKeyInfo), any CCMgr can detect whether a CCKeyInfo:CC_Key is set (although hidden) based on the CCKeyInfo:CC_KeyProtectBit. If the CCKeyInfo:CC_KeyProtectBit is set, the CCKeyInfo:CC_Key is set and hidden. Otherwise the return CCKeyInfo:CC_Key is the real one even if it is zero.

A10.4.1.1.6 LEVELS OF PROTECTION

There are four different protection levels based on the CC_Key, depending on the system requirements.

Table 497 Protection Levels

CC_Key	CC_KeyProtectBit	CC_KeyLeasePeriod	Description
0	any	any	No Protection provided. Any CCMgr can issue sets and gets
non-zero	0	n/a	Protection provided, but allows CCMgrs to read the CCKeyInfo:CC_Key in this mode
non-zero	1	non-zero	Protection provided and does not allow anyone to read the CC_Key in the node until the lease period has expired. The CC_Key lease period is a mechanism to allow the CC_Key to be protected only for a given amount of time.
non-zero	1	0	Protection provided and does not allow the CC_Key in the node to be read by other CCMgrs. It must be noted that if the lease period was set to 0 (infinite) and the CCMgr that set it is no longer operational, there is no possibility for other CCMgrs to ever read it. So if the CC_Key is not provided by some unspecified way to the other CCMgrs, the CCMgtA of this node will never be accessible again

A10.4.1.2 CONGESTION CONTROL LOG DATA

For the CongestionLog attribute, the payload is 224 bytes, starting at byte 32 and extending into the CCMgt Data field. For attributes other than CongestionLog, the contents of the Congestion Control Log Data field is reserved.

A10.4.1.3 CCMGT DATA

For attributes other than CongestionLog, the payload is 192 bytes, starting at byte 64.

A10.4.2 METHODS

The Congestion Control Class uses a subset of the common methods described in 13.4.5 Management Class Methods on page 721.

CA10-58: Congestion Control shall support the methods listed in Table 498 Congestion Control Methods on page 1653. All method type values not listed in the Table are reserved.

Table 498 Congestion Control Methods

Method Type	Value	Description
CCMgtGet()	0x01	Request (read) a class specific information attribute.
CCMgtSet()	0x02	Request (write) that a class specific information attribute be set.
CCMgtGetResp()	0x81	Response from a Get() or Set() request.
CCMgtTrap()	0x05	Unsolicited datagram sent to the Congestion Control Management entity. Contains the Notice Attribute as defined in A10.4.3.2 Traps and Notices on page 1655 to identify the trap.
CCMgtTrapRepress()	0x07	Block repetition of notification.

A10.4.3 ATTRIBUTES

CA10-59: A Congestion Control Management Agent shall support the attributes listed in Table 499 Congestion Control Attributes on page 1653. All attributes not listed in Table 499 Congestion Control Attributes on page 1653 are reserved.

The attributes are described in detail in the sections following the table.

Table 499 Congestion Control Attributes

Attribute Name	Attribute ID	Attribute Modifier	Description	Applicable to
ClassPortInfo	0x0001	0x00000000	Provides information about the CCMgt Agent. See A10.4.3.1 ClassPortInfo on page 1655.	Switch & CA

Table 499 Congestion Control Attributes (Continued)

Attribute Name	Attribute ID	Attribute Modifier	Description	Applicable to
Notice	0x0002	0x00000000	Provides Trap details. See A10.4.3.2 Traps and Notices on page 1655.	Switch & CA
CongestionInfo	0x0011	0x00000000	Provides information about the congestion capability of a device. See A10.4.3.3 CongestionInfo on page 1657.	Switch & CA
CongestionKeyInfo	0x0012	0x00000000	Provides CC_Key information for a port. Note: For a switch this is only applicable to port 0. See A10.4.3.4 CongestionKeyInfo on page 1657.	Switch & CA
CongestionLog	0x0013	0x00000000	Returns CA or switch congestion log. See A10.4.3.5 CongestionLog on page 1657.	Switch & CA
SwitchCongestionSetting	0x0014	0x00000000	Default buffer thresholds at which packet marking should occur, and device specific fields. See A10.4.3.6 SwitchCongestionSetting on page 1659.	Switch
SwitchPortCongestionSetting	0x0015	Block Identifier	Provides the trigger levels, packet_size, and credit return rate controls for a port. See A10.4.3.7 SwitchPortCongestionSetting on page 1661.	Switch
CACongestionSetting	0x0016	0x00000000	Provides rate control reduction and logging settings. See A10.4.3.8 CACongestionSetting on page 1662.	CA
CongestionControlTable	0x0017	Sequence Number	Indicates the starting element of the table for this request. See A10.4.3.9 CongestionControlTable on page 1664.	CA
TimeStamp	0x0018	0x00000000	Returns the current timer value from a device. See A10.4.3.10 TimeStamp on page 1666.	Switch & CA

Table 500 Congestion Control Attribute / Method Map

Attribute Name	CCMgtGet()	CCMgtSet()	CCMgtTrap()	CCMgtTraprepress
ClassPortInfo	X			
Notice	X	X	X	X
CongestionInfo	X			
CongestionKeyInfo	X	X		
CongestionLog	X			
SwitchCongestionSetting	X	X		
SwitchPortCongestionSetting	X	X		
CACongestionSetting	X	X		
CongestionControlTable	X	X		
TimeStamp	X			

A10.4.3.1 CLASSPORTINFO

CA10-60: A CCMgtA shall support ClassPortInfo as described in 13.4.8.1 ClassPortInfo on page 734.

The ClassPortInfo attribute is described in 13.4.8.1 ClassPortInfo on page 734. There are no Class-specific bits for the Congestion Control Class-PortInfo:CapabilityMask. See Table 501 Congestion Control ClassPort-Info:CapabilityMask on page 1655.

Table 501 Congestion Control ClassPortInfo:CapabilityMask

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.8.1 ClassPortInfo on page 734.
8	EnhancedPort0CC	Switch only: set if the EnhacedPort0 supports CA Congestion Control. (Note a switch can support Congestion control on data ports without supporting it on EnhancedPort0)
9-15	-	Reserved

A10.4.3.2 TRAPS AND NOTICES

Notice and Trap Queues allow a CCMgtA to inform a Congestion Manager of a CC_Key violation.

CA10-61: A Congestion Control Management Agent shall support either a Trap or a Notice Queue.

It is expected that a Congestion Manager will be capable of supporting both Trap and Notice Queues.

Traps and Notice Queues are described in detail in 13.4.9 Traps on page 741 and 13.4.10 Notice Queue on page 743. Congestion Control will use the common framework and this section serves only to describe behavior that is unique to the Congestion Control Class.

.

Table 502 Congestion Control Traps

Name	Type	Number	Details
CC_KeyViolation	Security	0x0000	See Table 503 Notice details for Trap 0x0000 CC_KeyViolation on page 1656

oA10-6: If the CCMgtA supports Traps, then the CCMgtA shall only use the Type and Number listed in Table 502 Congestion Control Traps on page 1655

Table 503 Notice details for Trap 0x0000 CC_KeyViolation

Component	Length (Bits)	Description
Source LID	16	Source Local Identifier, from the LRH of the offending MAD.
Method	8	Method, from Common MAD Header of the offending MAD.
Reserved	8	Reserved
Attribute ID	16	Attribute ID, from Common MAD Header of the offending MAD.
Attribute Modifier	32	Attribute Modifier, from Common MAD Header of the offending MAD.
Reserved	8	Reserved
QP	24	Destination Queue Pair number from the BTH of the offending MAD.
CC_Key	64	Congestion Control Key specified in the offending MAD.
Source GID	128	The Source Global Identifier from the GRH of the offending MAD. If no GRH is present in the offending packet, this field shall be binary zeroes.
Padding	192	Shall be ignored on read. Content is unspecified.

oA10-7: If the CCMgtA supports Traps as indicated in ClassPortInfo, then the CCMgtA shall set the CC_KeyViolation notice as specified in Table 503 Notice details for Trap 0x0000 CC_KeyViolation on page 1656 Fields shall be filled with the information corresponding to the description of the trap.

oA10-8: If the CCMgtA supports Notices as indicated in ClassPortInfo, then the CCMgtA shall post the CC_KeyViolation notice as specified in Table 503 Notice details for Trap 0x0000 CC_KeyViolation on page 1656 to the notice queue when a CC_Key mismatch is detected.

oA10-9: The notice shall only be visible on the port that detected the CC_Key mismatch.

A10.4.3.3 CONGESTIONINFO

The CongestionInfo attribute provides the Congestion Control Manager with information on the capabilities of the node. This attribute is supported by both CAs and switches.

Table 504 CongestionInfo

Component	Access	Length (bits)	Offset (bits)	Description
CongestionInfo	RO	16	0	• Bit0 = 1: Supports Credit Starvation (switch only) • Bit0 = 0: Does not support Credit Starvation All other bits are reserved
ControlTableCap	RO	8	16	Number of 64 entry blocks in the CongestionControlTable. At least two 64 entry blocks must be supported. (CA only)

A10.4.3.4 CONGESTIONKEYINFO

The CongestionKeyInfo attribute is used by the Congestion Control Manager to set and retrieve the node's CC_Key and protection properties.

Table 505 CongestionKeyInfo

Component	Access	Length (bits)	Offset (bits)	Description
CC_Key	RW	64	0	The 8-byte CC_Key used in all CongestionControl MADs by all valid CCMgrs. A value of 0 means that no CC_Key check is ever performed by the CCMgtA.
CC_KeyProtectBit	RW	1	64	See A10.4.1.1.6 Levels of Protection on page 1652 for details.
Reserved	RW	15	65	Reserved
CC_KeyLeasePeriod	RW	16	80	Timer value used to indicate how long the CC_KeyProtectBit is to remain non zero after a CCMgtSet(CCKeyInfo) MAD that failed a CC_Key check is dropped. The value of the timer indicates the number of seconds for the lease period. With a 16 bit counter, the period can range from 1 second to approximately 18 hours. 0 shall mean infinite.
CC_KeyViolations	RO	16	96	Number of MADs that have been received at this node since power-on or reset that have been dropped due to a failed CC_Key check if such a counter is implemented. Otherwise it shall be 0xFFFF. When the CCMgr sets this component to 0x0000, the counter is reset to 0x0000 and counting resumes. Setting the counter to a value other than zero results in the counter being left unchanged.

A10.4.3.5 CONGESTIONLOG

The CongestionLog attribute provides the Congestion Control Manager with information on congestion events detected at congestion control ca-

pable switches and CAs. For switches, it returns a map of ports that marked packets within a specified time window. For CAs, it returns a map of SLs that have hit their threshold within a specified time window.

Table 506 CongestionLog (switch)

Component	Access	Length (bits)	Offset (bits)	Description
Logtype	RO	8	0	= 0x1 Switch
CongestionFlags	RO	8	8	Bit 0 = 1, CC_Key lease period timer active. Bit 0 = 0, CC_Key lease period timer inactive. All other bits are reserved.
LogEventsCounter	RO	16	16	Number of CongestionLogEvents since log last sent
CurrentTimeStamp	RO	32	32	32 bits wraps ~1hr with a 1.024 µsec tick
PortMap	RO	256	64	If a bit is set to 1 then the corresponding port has marked packets with a FECN. Bit 0: reserved Bit 1: port 1 ... Bit 254: port 254 Bit 255: reserved
CongestionEntryList	RO	1440	320	Contains an array of the twenty most recent CongestionLogEvent entries. (See Table 507 on page 1658.)

Table 507 CongestionLogEvent (switch)

Component	Access	Length (bits)	Offset (bits)	Description
SLID	RO	16	0	Source LID of congestion event
DLID	RO	16	16	Destination LID of congestion event
SL	RO	4	32	Service level of congestion event
Reserved	RO	4	36	reserved
Timestamp	RO	32	40	Timestamp of congestion event

Table 508 CongestionLog (CA)

Component	Access	Length (bits)	Offset (bits)	Description
LogType	RO	8	0	= 0x2 CA

Table 508 CongestionLog (CA) (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
CongestionFlags	RO	8	8	Bit 0 = 1, CC_Key lease period timer active. Bit 0 = 0, CC_Key lease period timer inactive. All other bits are reserved.
ThresholdEventCounter	RO	16	16	Number of events since log last sent.
ThresholdCongestion-EventMap	RO	16	32	Contains an array of sixteen bits, one for each SL. • If set to 1b, at least one event has occurred on this SL before the timestamp wrapped twice. • If set to 0b, no events have occurred on this SL for a period equal to twice the maximum timestamp.
CurrentTimeStamp	RO	16	48	Timestamp when log sent
CongestionLogEvent	RO	1664	64	Contains an array of the sixteen most recent CongestionLogEvent entries. (See Table 509 on page 1659.)

Table 509 CongestionLogEvent (CA)

Component	Access	Length (bits)	Offset (bits)	Description
Local_QP_CN_Entry	RO	8	0	Local QP that reached CN Threshold. Set to zero if port threshold reached
Remote_QP_Number_CN_Entry	RO	24	8	Remote QP that is connected to local QP. Set to zero for datagram QPs.
SL_CN_Entry	RO	4	32	Service Level associated with local QP
Service_Type_CN_Entry	RO	4	36	Service Type of local QP • b'0000' - RC • b'0001' - UC • b'0010' - RD • b'0011' - UD
Remote_LID_CN_Entry	RO	32	40	LID of remote port that is connected to local QP. Set to zero for datagram service.
Timestamp_CN_Entry	RO	32	72	Timestamp when threshold reached.

A10.4.3.6 SWITCHCONGESTIONSETTING

The Congestion Control Manager uses the SwitchCongestionSetting attribute to program and retrieve the switch's congestion settings. This allows the CCMgr to control such attributes as how aggressively the switch

marks congested packets, whether packets which are the victim of congestion are marked, and credit starvation.

Table 510 SwitchCongestionSetting

Component	Access	Length (bits)	Offset (bits)	Description
Control_Map	RW	32	0	Indicates which components of this attribute are valid • bit 0: a 1 indicates that the victim mask is valid. • bit 1: a 1 indicates that the credit mask is valid. • bit 2: a 1 indicates that the Threshold and Packet_Size components are valid. • bit 3: a 1 indicates that the CS_threshold and CS_ReturnDelay components are valid. • bit 4: a 1 indicates that the Marking_Rate is valid. all other bits reserved.
Victim_Mask	RW	256	32	If the bit set to 1, then the port corresponding to that bit shall mark packets that encounter congestion with a FECN, whether they are the source or victim of congestion. (See A10.2.1.1.1 Root vs. Victim on page 1638.) • bit 0: port 0 (enhanced port 0 only) • ... • bit 254: port 254 • bit 255: reserved
Credit_Mask	RW	256	288	If bit set to 1, then the port corresponding to that bit shall apply Credit Starvation. • bit 0: port 0 (enhanced port 0 only) • ... • bit 254: port 254 • bit 255: reserved
Threshold	RW	4	544	A value that indicates how aggressive congestion marking should be: • 0x0: No packet marking • 0x1: Loose • 0x2: • ... • 0xF: Very aggressive (See A10.2.1.1 Congestion Detection on page 1636.)
Reserved	RW	4	548	reserved
Packet_Size	RW	8	552	Any packet less than this size will not be marked with a FECN. This fields units are credits, a value greater than the maximum packet size will result in no packets being marked with a FECN.

Table 510 SwitchCongestionSetting (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
CS_Threshold	RW	4	560	• 0x0: No credit starvation • 0x1: Loose • 0x2: • .. • 0xF: Very aggressive see A10.2.1.1 Congestion Detection on page 1636
Reserved	RW	4	564	reserved
CS_ReturnDelay	RW	16	568	The same format as a CCT entry, that controls the credit rate return when active. See A10.2.2.1.1 CCT Entry Format on page 1643
Marking_Rate	RW	16	584	The value that provides the mean number of packets, between marking eligible packets with a FECN.

This may be used to set the Victim and/or Credit Starvation (CS) Marker on each of the ports, A seting of the Marker in each case will activate that functionality on that port. It shall also be used to set a default setting for Threshold, packet_size, CS _threshold, and CS_ReturnRate, along with the packet marking rate to the same value on all ports. This is the only way of setting the marking rate for a switch. The control map will indicate which attributes are active. A CongestionGet() shall return the latest values set in a device for each active attribute set in the Control_Map.

All attributes apply to both data ports and port zero, as it is only a switch that sets a FECN in a packet, except for CS_Threshold and CS_ReturnDelay, which applies to data ports and enhanced port 0 only.

A10.4.3.7 SWITCHPORTCONGESTIONSETTING

The Congestion Control Manager uses the SwitchPortCongestionSetting attribute to program and retrieve specific switch port's congestion settings.

Table 511 SwitchPortCongestionSetting Attribute

Component	Access	Length(bits)	Description
SwitchPortCongestionSetting Block	RW	1536	Contains an array of 64 SwitchPortCongestionSettingElements. (See Table 512 on page 1661)

Table 512 SwitchPortCongestionSettingElement

Component	Length (bits)	Offset (bits)	Description
Valid	1	0	When set to 1, indicates this switch port congestion setting element is valid.

Table 512 SwitchPortCongestionSettingElement (Continued)

Component	Length (bits)	Offset (bits)	Description
Control_Type	1	1	Indicates which type of attribute is being set: • 0b = Congestion Control parameters are being set. • 1b = Credit Starvation parameters are being set.
Reserved	2	2	reserved
Threshold	4	4	• When Control_Type=0b, contains the congestion threshold value (Threshold) for this port. • When Control_Type=1b, contains the credit starvation threshold (CS_Threshold) value for this port. • 0x0: None • 0x1: Loose • 0x2: • ... • 0xF: Very aggressive See A10.2.1.1 Congestion Detection on page 1636.
Cong_Parm	16	8	• When Control_Type = 0b, this field contains the minimum size of packets (Packet_Size) that may be marked with a FECN. Packets smaller than this size will not be marked. The packet size is specified in credits. • When Control_Type = 1b, this field contains the congestion control return delay (CS_ReturnDelay). The delay is specified in the same format as a CCT entry. (See A10.2.2.1.1 CCT Entry Format on page 1643.

The AttributeModifier is a pointer to a block of 64 Congestion controls for ports to which this attribute applies. Valid values are from 0 to 3, and are further limited by the number of ports within a switch.

Any entries in the SwitchPortCongestionSetting Block beyond the number of ports within a switch will be ignored on write and read back as zero

A10.4.3.8 CACONGESTIONSETTING

The Congestion Control Manager uses the CACongestionSetting attribute to program and retrieve congestion settings on channel adapters.

Table 513 CACongestionSetting

Component	Access	Length (bits)	Offset (bits)	Description
Port_Control	RW	16	0	Specifies congestion attributes for this port: • bit 0 = 0; QP based congestion control • bit 0 = 1; SL/Port based congestion control All other bits are reserved.

Table 513 CACongestionSetting (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
Control_Map	RW	16	16	An array of sixteen bits, one for each SL. Each bit indicates whether or not the corresponding SL entry is to be modified. If 1b, then the values for the corresponding SL must be set.
CACongestionEntryList	RW	640	32	Specifies a list of sixteen CACongestionEntries, one per service level. (See Table 514 on page 1664.)

A10.4.3.8.1 PORT_CONTROL

Port_Control indicates whether congestion control is on an SL/port or a QP basis.

If it is on a QP basis, each QP on a port will have a unique CCTI, so BECNs received on one QP will have no effect on other QPs on that same SL.

If it is on an SL/port basis, all QPs on that same SL, and behind the same port, will share a common CCTI, so that BECNs received on one QP, will effect the IRD of all QPs on that SL/port.

A10.4.3.8.2 CONTROL_MAP

The Control_Map identifies which of the sixteen SLs, will be set by this attribute. When set to 1b, the corresponding SL will have its CCTI_Increase, CCTI_Timer, Trigger_Threshold and CCTI_Min set from the corresponding entries in this attribute.

A10.4.3.8.3 CACONGESTIONENTRYLIST

The CACongestionEntryList is a list of sixteen CACongestionEntries. Each entry corresponds to a service level, from zero to fifteen.

All sixteen entries must be specified in the attribute, however only entries with their corresponding Control_Map bit equal to 1b, will be used. Entries with their Control_Map bit equal to 0b are ignored.

A10.4.3.8.4 CACONGESTIONENTRY

Table 514 CACongestionEntry

Component	Access	Length (bits)	Offset (bits)	Description
CCTI_Increase	RW	8	0	The number to be added to the table Index (CCTI) on the receipt of a BECN.
CCTI_Timer	RW	16	8	When the timer expires it will be reset to its specified value, and 1 will be decremented from the CCTI.
Trigger_Threshold	RW	8	24	When the CCTI is equal to this value, an event is logged in the CA's cyclic event log.
CCTI_Min	RW	8	32	The minimum value permitted for the CCTI. This is used to impose a minimum injection rate delay on the SL.

A10.4.3.9 CONGESTIONCONTROLTABLE

The Congestion Control Manager uses the CongestionControlTable attribute to program and retrieve entries in a channel adapter's Congestion Control Table. There is a separate Congestion Control Table for each port.

In the case of a switch with enhanced port 0, the "device" beyond the port may be considered a CA, and this attribute may be used to program its congestion control table.

Table 515 CongestionControlTable

Component	Access	Length (bits)	Offset (bits)	Description
CCTI_Limit	RW	16	0	This value indicates the maximum valid CCTI for this table. See A10.2.2.1.2 Rate Decrease on page 1645.
CCTI_Entry_List	RW	1024	16	Specifies a list of up to 64 CongestionControlTableEntries. (See Table 516 on page 1666.)

A10.4.3.9.1 CCTI_LIMIT

Specifies the maximum value that the CCTI may reach. It is the number of valid (i.e. defined) CCT entries plus one. The last CCTI_Limit specified in a MAD will be the one used by the CA.

Note: To determine the maximum number of entries the CA supports, check the ControlTableCap component of the CongestionInfo attribute. (See A10.4.3.3 CongestionInfo on page 1657.)

Since a congestion control table may support far more than 64 entries, yet only 64 may be defined in a single MAD, it is suggested that each MAD specify a CCTI_Limit which will not allow the CCTI to extend beyond the valid entries defined up to that point.

So for example, if you were defining a table with 150 entries, the first MAD sent defines the first 64 entries, and CCTI_Limit is set to 63; the second MAD would contain the next 64 entries, and CCTI_Limit set to 127 and the final MAD would contain the remaining 22 entries, and CCTI_Limit set to 149. In this way, if BECNs were being received while the table is being defined, it would not run off the end of the table.

If the tables are reloaded and the resulting CCTI_Limit is found to be lower than the current CCTI, the CCTI will be set to the CCTI_Limit. This can be done on a case by case basis, and does not require the CA to check all its CCTI's when a CCTI_Limit is lowered.

A10.4.3.9.2 CCTI_ENTRY_LIST

Specifies a list of up to 64 CongestionControlTableEntries. The origin of these table entries is equal to 64 * attribute modifier + 1. In other words, if the attribute modifier is 0, then entries 1 - 64 are being defined. If the attribute modifier is 1, then entries 65 - 128 are being defined.

Note: Since these MADs are directed to specific ports, the attribute modifier is not required to be used to identify the port.

Only entries that are within the range of the CCTI_Limit specified in the attribute need be specified. In other words, if the CCTI_Limit is 149, and the attribute modifier is 2 (i.e. entries 129 - 192), then the MAD may be truncated to contain only 22 entries.

A10.4.3.9.3 CONGESTIONCONTROLTABLEENTRY

Table 516 CongestionControlTableEntry

Component	Access	Length (bits)	Offset (bits)	Description
CCT_Shift	RW	2	0	This is the shft value used when calculating the injection rate delay. (See A10.2.2.1.1 CCT Entry Format on page 1643.)
CCT_Multiplier	RW	14	2	This is the multiplier used when calculating the injection rate delay (See A10.2.2.1.1 CCT Entry Format on page 1643.).

A10.4.3.10 TIMESTAMP

The TimeStamp attribute allows the Congestion Control Manager to read a device's free running timer. This may be used to help synchronize events from different devices.

Table 517 TimeStamp

Component	Access	Length (bits)	Offset (bits)	Description
TimeStamp	RO	32	0	Free running clock that provides relative time information for a device. Time is kept in 1.024 µsec units.

A10.5 CONGESTION MANAGEMENT PERFORMANCE COUNTERS

These performance counters may be accessed though a collecting or polling mechanism. Mechanisms for sampling counters are described in 16.1 Performance Management on page 930.

The following describes the additions to the Component Mask, and other fields that are currently reserved in the version 1.1 of the specification.

A10.5.1 PORTSAMPLESCONTROL

The following are additions to the PortSamplesControl:OptionMask in order to enable a device to disclose the congestion management performance counters that it supports. (Table 222 PortSamplesControl on page 934)

Table 518 Addition to PortSamplesControl

Component	Access	Length (bits)	Offset (bits)	Description
OptionMask	RO	64	96	Bit 49 = PortRcvConCtrl:PortPktRcvFECN Bit 50 = PortRcvConCtrl:PortPktRcvBECN Bit 51 = PortSLRcvFECN:PortSLPktRcvFECN[n] Bit 52 = PortSLRcvBECN:PortSLPktRcvBECN[n] Bit 53 = PortXmitConCtrl:PortXmitTimeCong Bit 54 = PortVLXmitTimeCong:PortVLXmitTimeCong[n]

A10.5.2 COUNTER SELECT VALUES

The following are additions to the CounterSelect Values in order to support the Congestion Control performance counters. (see Table 223 CounterSelect Values on page 941)

Table 519 Additional CounterSelect Values

Sample Select Value	Name	Description
0x5001	PortRcvConCtrl:PortPktRcvFECN	See Table 522 PortRcvConCtrl on page 1669
0x5002	PortRcvConCtrl:PortPktRcvBECN	See Table 522 PortRcvConCtrl on page 1669
0x5n03	PortSLRcvFECN:PortSLPktRcvFECN[n]	See Table 523 PortSLRcvFECN on page 1670
0x5n04	PortSLRcvBECN:PortSLPktRcvBECN[n]	See Table 524 PortSLRcvBECN on page 1671
0x5005	PortXmitConCtrl:PortXmitTimeCong	See Table 525 PortXmitConCtrl on page 1673
0x5n06	PortVLXmitDataRoot:PortVLXmitTimeCong[n]	See Table 526 PortVLXmitTimeCong on page 1673

A10.5.3 OPTIONAL PERFORMANCE MANAGEMENT ATTRIBUTES

The following are additions to the optional performance management attributes in order to support the Congestion Control performance metrics. (see Table 226 Optional Performance Management Attributes on page 950).

Table 520 Additional Optional Performance Management Attributes

Attribute Name	Attribute ID	Attribute Modifier	Description
PortRcvConCtrl	0x0031	0x00000000	See A10.5.4 PortRcvConCtrl on page 1668
PortSLRcvFECN	0x0032	0x00000000	See A10.5.5 PortSLRcvFECN on page 1669

Table 520 Additional Optional Performance Management Attributes (Continued)

Attribute Name	Attribute ID	Attribute Modifier	Description
PortSLRcvBECN	0x0033	0x00000000	See A10.5.6 PortSLRcvBECN on page 1671
PortXmitConCtrl	0x0034	0x00000000	See A10.5.7 PortXmitConCtrl on page 1673
PortVLXmitTimeCong	0x0035	0x00000000	See A10.5.8 PortVLXmitTimeCong on page 1673

The following are additions to the optional performance management attribute / method map in order to support the congestion control performance metrics. (see Table 227 Optional Performance Management Attribute / Method Map on page 951).

Table 521 Additional Optional Performance Management Attribute / Method Map

Attribute Name	PerformanceGet( )	PerformanceSet( )
PortRcvConCtrl	X	X
PortSLRcvFECN	X	X
PortSLRcvBECN	X	X
PortXmitConCtrl	X	X
PortVLXmitTimeCong	X	X

A10.5.4 PORTRCVCONCTRL

The PortRcvConCtrl provides a mechanism for counting the number of packets received by a CA marked with a FECN or BECN. This counter is only provided by a congestion-control-capable CA.

Table 522 PortRcvConCtrl

Component	Access	Length (bits)	Offset (bits)	Description
Reserved	RO	8	0	Reserved
PortSelect	RW	8	8	Selects the port, as defined in Table 222 PortSamplesControl on page 934, for which the statistics are reported. Statistics are accumulated for all SL's on a port. If gathering data from all ports at once is supported (see Table 221 PerformanceManagement ClassPortInfo:CapabilityMask on page 933), setting PortSelect to 0xFF will cause data from all valid ports to be accumulated. When Selecting invalid port values, any results are undefined.
CounterSelect	RW	16	16	When writing (Set), selects which counters are overwritten by the values specified in their respective fields. When reading (Get), this is ignored. Bit 0 - PortPktRcvFECN Bit 1 - PortPktRcvBECN
PortPktRcvFECN	RW	32	32	Total number of packets received at the port that contain a FECN mark in the BTH.
PortPktRcvBECN	RW	32	64	Total number of packets received at the port that contain a BECN mark in the BTH.

A10.5.5 PORTSLRCVFECN

The PortSLRcvFECN provides a mechanism for counting the number of packets received by a CA marked with a FECN on each SL. This counter is only provided by a congestion-control-capable CA.

Table 523 PortSLRcvFECN

Component	Access	Length (bits)	Offset (bits)	Description
Reserved	RO	8	0	Reserved
PortSelect	RW	8	8	Selects the port, as defined in Table 222 PortSamplesControl on page 934, for which the statistics are reported. If gathering data from all ports at once is supported (see Table 221 PerformanceManagement ClassPortInfo:CapabilityMask on page 933), setting PortSelect to 0xFF will cause data from all valid ports to be accumulated. When Selecting invalid port values, any results are undefined.
CounterSelect	RW	16	16	When writing (Set), selects which counters are overwritten by the values specified in their respective fields. When reading (Get), this is ignored. Bit 0 - PortSLRcvFECN0 Bit 1 - PortSLRcvFECN1 Bit 2 - PortSLRcvFECN2 Bit 3 - PortSLRcvFECN3 Bit 4 - PortSLRcvFECN4 Bit 5 - PortSLRcvFECN5 Bit 6 - PortSLRcvFECN6 Bit 7 - PortSLRcvFECN7 Bit 8 - PortSLRcvFECN8 Bit 9 - PortSLRcvFECN9 Bit 10 - PortSLRcvFECN10 Bit 11 - PortSLRcvFECN11 Bit 12 - PortSLRcvFECN12 Bit 13 - PortSLRcvFECN13 Bit 14 - PortSLRcvFECN14 Bit 15 - PortSLRcvFECN15
PortSLRcvFECN0	RW	32	32	Number of packets received on SL0 at the port that contain a FECN mark in the BTH.
PortSLRcvFECN1	RW	32	64	Similar count for SL1
PortSLRcvFECN2	RW	32	96	Similar count for SL2
PortSLRcvFECN3	RW	32	128	Similar count for SL3
PortSLRcvFECN4	RW	32	160	Similar count for SL4
PortSLRcvFECN5	RW	32	192	Similar count for SL5
PortSLRcvFECN6	RW	32	224	Similar count for SL6
PortSLRcvFECN7	RW	32	256	Similar count for SL7
PortSLRcvFECN8	RW	32	288	Similar count for SL8

Table 523 PortSLRcvFECN (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
PortSLRcvFECN9	RW	32	320	Similar count for SL9
PortSLRcvFECN10	RW	32	352	Similar count for SL10
PortSLRcvFECN11	RW	32	384	Similar count for SL11
PortSLRcvFECN12	RW	32	416	Similar count for SL12
PortSLRcvFECN13	RW	32	448	Similar count for SL13
PortSLRcvFECN14	RW	32	480	Similar count for SL14
PortSLRcvFECN15	RW	32	512	Similar count for SL15

A10.5.6 PORTSLRCVBECN

The PortSLRcvBECN provides a mechanism for counting the number of packets received by a CA marked with a BECN on each SL. This counter is only provided by a congestion-control-capable CA.

Table 524 PortSLRcvBECN

Component	Access	Length (bits)	Offset (bits)	Description
Reserved	RO	8	0	Reserved
PortSelect	RW	8	8	Selects the port, as defined in Table 222 PortSamplesControl on page 934, for which the statistics are reported. If gathering data from all ports at once is supported (see Table 221 PerformanceManagement ClassPortInfo:CapabilityMask on page 933), setting PortSelect to 0xFF will cause data from all valid ports to be accumulated. When Selecting invalid port values, any results are undefined.

Table 524 PortSLRcvBECN (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
CounterSelect	RW	16	16	When writing (Set), selects which counters are overwritten by the values specified in their respective fields. When reading (Get), this is ignored. Bit 0 - PortSLRcvBECN0 Bit 1 - PortSLRcvBECN1 Bit 2 - PortSLRcvBECN2 Bit 3 - PortSLRcvBECN3 Bit 4 - PortSLRcvBECN4 Bit 5 - PortSLRcvBECN5 Bit 6 - PortSLRcvBECN6 Bit 7 - PortSLRcvBECN7 Bit 8 - PortSLRcvBECN8 Bit 9 - PortSLRcvBECN9 Bit 10 - PortSLRcvBECN10 Bit 11 - PortSLRcvBECN11 Bit 12 - PortSLRcvBECN12 Bit 13 - PortSLRcvBECN13 Bit 14 - PortSLRcvBECN14 Bit 15 - PortSLRcvBECN15
PortSLRcvBECN0	RW	32	32	Number of packets received on SL0 at the port that contain a BECN mark in the BTH.
PortSLRcvBECN1	RW	32	64	Similar count for SL1
PortSLRcvBECN2	RW	32	96	Similar count for SL2
PortSLRcvBECN3	RW	32	128	Similar count for SL3
PortSLRcvBECN4	RW	32	160	Similar count for SL4
PortSLRcvBECN5	RW	32	192	Similar count for SL5
PortSLRcvBECN6	RW	32	224	Similar count for SL6
PortSLRcvBECN7	RW	32	256	Similar count for SL7
PortSLRcvBECN8	RW	32	288	Similar count for SL8
PortSLRcvBECN9	RW	32	320	Similar count for SL9
PortSLRcvBECN10	RW	32	352	Similar count for SL10
PortSLRcvBECN11	RW	32	384	Similar count for SL11
PortSLRcvBECN12	RW	32	416	Similar count for SL12
PortSLRcvBECN13	RW	32	448	Similar count for SL13
PortSLRcvBECN14	RW	32	480	Similar count for SL14
PortSLRcvBECN15	RW	32	512	Similar count for SL15

A10.5.7 PORTXMITCONCTRL

The PortXmitConCtrl provides a mechanism for counting the amount of time a switch port is in the congested state. This counter is only provided by a congestion-control-capable switch.

Table 525 PortXmitConCtrl

Component	Access	Length (bits)	Offset (bits)	Description
Reserved	RO	8	0	Reserved
PortSelect	RW	8	8	Selects the port, as defined in Table 222 PortSamplesControl on page 934, for which the statistics are reported. Statistics are accumulated for all SL's on a port. If gathering data from all ports at once is supported (Table 221 Performance Management ClassPortInfo:CapabilityMask on page 933), setting PortSelect to 0xFF will cause data from all valid ports to be accumulated. When Selecting invalid port values, any results are undefined.
CounterSelect	RW	16	16	When writing (Set), selects which counters are overwritten by the values specified in their respective fields. When reading (Get), this is ignored. Bit 0 - PortXmitTimeCong
PortXmitTimeCong	RW	32	32	Total number of ticks during which the port selected by PortSelect was in the congested state on any VL.

For a definition of a tick, see Table 222 PortSamplesControl on page 934.

A10.5.8 PORTVLXMITTIMECONG

The PortVLXmitTimeCong provides a mechanism for counting the amount of time a switch port data VL is in the congested state. This counter is only provided by a congestion-control-capable switch.

Table 526 PortVLXmitTimeCong

Component	Access	Length (bits)	Offset (bits)	Description
Reserved	RO	8	0	Reserved
PortSelect	RW	8	8	Selects the port, as defined in Table 222 PortSamplesControl on page 934, for which the statistics are reported. If gathering data from all ports at once is supported (see Table 221 Performance Management ClassPortInfo:CapabilityMask on page 933), setting PortSelect to 0xFF will cause data from all valid ports to be accumulated. When Selecting invalid port values, any results are undefined.

Table 526 PortVLXmitTimeCong (Continued)

Component	Access	Length (bits)	Offset (bits)	Description
CounterSelect	RW	16	16	When writing (Set), selects which counters are overwritten by the values specified in their respective fields. When reading (Get), this is ignored. Bit 0 - PortVLXmitTimeCong0 Bit 1 - PortVLXmitTimeCong1 Bit 2 - PortVLXmitTimeCong2 Bit 3 - PortVLXmitTimeCong3 Bit 4 - PortVLXmitTimeCong4 Bit 5 - PortVLXmitTimeCong5 Bit 6 - PortVLXmitTimeCong6 Bit 7 - PortVLXmitTimeCong7 Bit 8 - PortVLXmitTimeCong8 Bit 9 - PortVLXmitTimeCong9 Bit 10 - PortVLXmitTimeCong10 Bit 11 - PortVLXmitTimeCong11 Bit 12 - PortVLXmitTimeCong12 Bit 13 - PortVLXmitTimeCong13 Bit 14 - PortVLXmitTimeCong14 Bit 15 - Reserved
PortVLXmitTimeCong0	RW	32	32	Total number of ticks during which the port selected by PortSelect was in the congested state on VL0.
PortVLXmitTimeCong1	RW	32	64	Similar count for VL1
PortVLXmitTimeCong2	RW	32	96	Similar count for VL2
PortVLXmitTimeCong3	RW	32	128	Similar count for VL3
PortVLXmitTimeCong4	RW	32	160	Similar count for VL4
PortVLXmitTimeCong5	RW	32	192	Similar count for VL5
PortVLXmitTimeCong6	RW	32	224	Similar count for VL6
PortVLXmitTimeCong7	RW	32	256	Similar count for VL7
PortVLXmitTimeCong8	RW	32	288	Similar count for VL8
PortVLXmitTimeCong9	RW	32	320	Similar count for VL9
PortVLXmitTimeCong10	RW	32	352	Similar count for VL10
PortVLXmitTimeCong11	RW	32	384	Similar count for VL11
PortVLXmitTimeCong12	RW	32	416	Similar count for VL12
PortVLXmitTimeCong13	RW	32	448	Similar count for VL13
PortVLXmitTimeCong14	RW	32	480	Similar count for VL14

A10.6 COMPLIANCE SUMMARY

This annex specifies two new compliance categories (see Chapter 20: Volume 1 Compliance Summary on page 1072 for an explanation of compliance categories and qualifiers). These new categories are:

- CCMgt Switch
- CCMgt CA

A10.6.1 CCMGT SWITCH COMPLIANCE CATEGORY

In order to claim conformance to the InfiniBand Architecture Specification for the Compliance Category of CCMgt Switch, a product shall meet all the requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support. The TRAP and NOTICE compliance qualifiers defined in chapter 20 apply for this category.

There is one further Compliance Qualifier for the CCMgt Switch. It is:

- SCS - A switch supporting Switch Credit Starvation

A10.6.2 CCMGT CA COMPLIANCE CATEGORY

In order to claim conformance to the InfiniBand Architecture Specification for the Compliance Category of CCMgt CA, a product shall meet all the requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support. The TRAP and NOTICE compliance qualifiers defined in chapter 20 apply for this category.