COMPUTER NETWORKS
CS 45201
CS 55201

CHAPTER 2
Data Link Networks

P. Farrell / H. Peyravi

Department of Computer Science
Kent State University
Kent, Ohio 44242
farrell@cs.kent.edu
http://www.cs.kent.edu/~ peyravi

Fall 2001

Contents

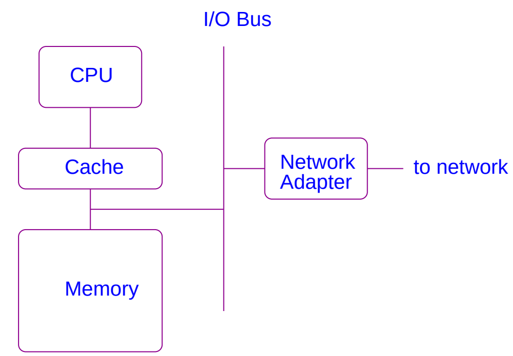
Hardware Building Blocks

Network Connecting Problems

- Physical connection (coax, fiber, ...)
- Encoding/Decoding data bits.
- Framing, packets, messages.
- Error detection.
- Reliable delivery despite errors.
- Media Access Control (MAC).
  $\Longrightarrow$ These issues are implemented in the network adaptor (board).
- $\Longrightarrow$ We will study the above problems in the context of
  - Point-to-Point links
  - Carrier Sense Multiple Access, CSMA networks (Ethernet)
  - Token Rings, Fiber Distributed Data Interface

Network Nodes

- Assume a general-purpose (programmable) computer; with special-purpose hardware.

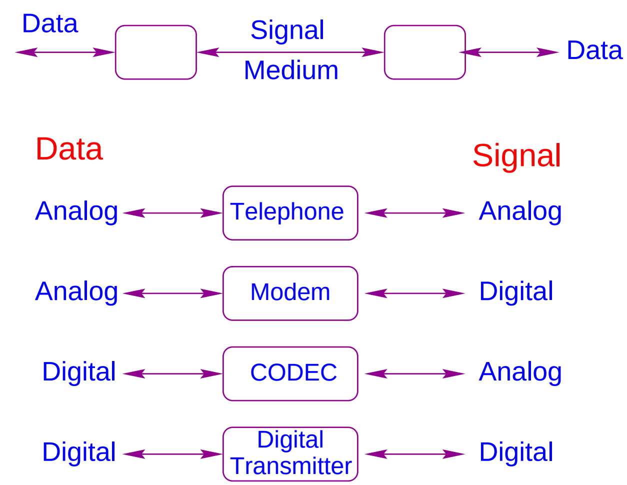


- A *device driver* manages the adaptor
- Finite memory (implies limited buffer space)
- Connects to network via a *network adaptor*
- Fast processor, slow memory

Network Links

- Links propagate signals
  - Analog: continuous
  - Digital: discrete
- Binary data are *encoded* in to
  - Analog signals: *modulator (modem)*
  - Digital signals: *demodulator*
- A digital *transmitter* transmits binary data over a digital link.
- *full duplex* links
- *half duplex* links

Data vs Signal



Some Physical Medium

Type	Speed	Distance
Category 5 twisted pair	10-100Mbps	100m
50-ohm coax (ThinNet)	10-100Mbps	200m
75-ohm coax (ThickNet)	10-100Mbps	500m
Multimode fiber	100Mbps	2km
Single-mode fiber	100-2400Mbps	40km

- Can be leased or owned

Standard Links

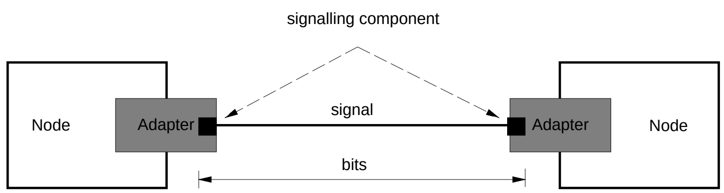
Type	Bandwidth	Applications
ISDN	64 Kbps	for digital voice/data
T1	1.544 Mbps	24 64Kbps, old technology
T3	44.736 Mbps	30 T1
STS-1	51.840 Mbps	sync. transfer signal optical
STS-3	155.250 Mbps	for optical fiber
STS-12	622.080 Mbps	for optical fiber
STS-24	1.244160 Gbps	for optical fiber
STS-48	2.488320 Gbps	for optical fiber

- The device that encodes analog voice into digital ISDN link is called *CODEC* (coder/decoder).
- STS-N links are sometimes called OC-N (optical carrier).
- STS-N is used for *electrical* device connected to the link.
- OC-N is used for *optical* device connected to the link.

Encoding

Overview

- Signals propagate over a physical medium.
  - Digital signals
  - Analog signals
- Data can be either digital or analog; we're interested in digital data.
- Problem: Encode the binary data that the source node wants to send to the destination node into the signal that propagates over

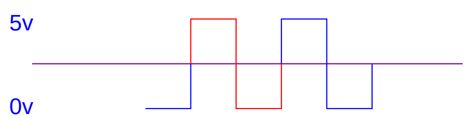


Maximum Data Rate of a Channel

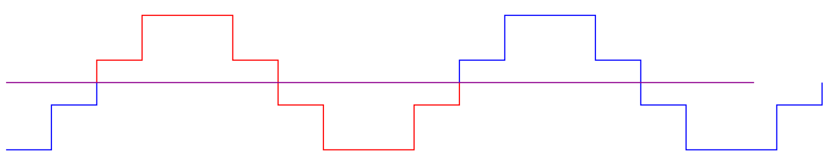
- Nyquist (1924) stated that for a noise-free channel with bandwidth $W(Hz)$, and multilevel signaling M, the capacity (bps) can be computed as

$$C = 2W \ \log_2 M$$

- Doubling W doubles the data rate.
- The presence of noise can corrupt one or more bits. If data rate is increased, the bits become shorter , and more bits are affected by a given noise pattern.
- At a given noise level, the higher the data rate, the higher the error rate.



Bilevel Encoding



Multilevel Encoding (M=4)

Shannon's Theorem

- Shannon (1948) developed a formula to identify the upper bound on the channel capacity.
  - The signal-to-noise ratio (S/N) is the ratio of power in a signal to the power contained in the noise that is present at a particular point in the transmission.

$$S/N = 10 \ \log_{10} \frac{signal\ power}{noise\ power}$$

  - The maximum channel capacity is computed as

$$C = B \ \log_2(1 + \frac{S}{N})$$

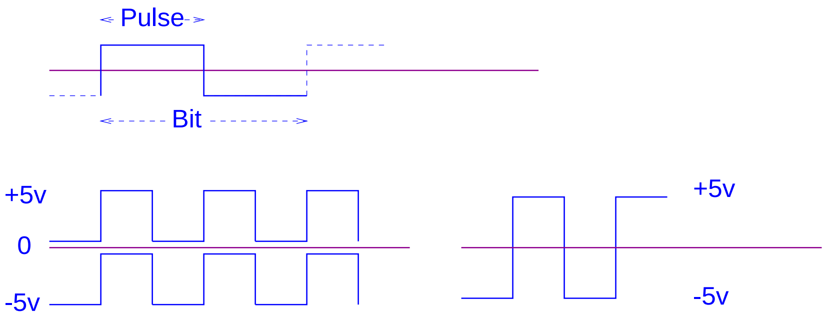
    where C is the capacity in bits per second and B is the bandwidth in Hz.
- For example a noiseless 3-kHz channel cannot transmit binary signals at a rate exceeding 6000 bps.
- A channel of 3000-Hz bandwidth, and a signal to thermal noise of 30 dB can never transmit more than 30,000 bps.

$$= 30\text{dB} = 10 \ \log_{10}(S/N)$$

$$S/N = 1000$$

$$C = 3000 \ \log_2(1 + 1000) = 3000 \times 9.9673 < 30000\text{bps}$$

Coding Terminology



- Signal element: Pulse
- Modulation Rate: $\frac{1}{\text{Duration of the smallest element}}$=Baud rate
- Data Rate: Bits per second
- Data Rate is a function of
  - bandwidth
  - signal/noise ratio
  - encoding technique

Transmission Media

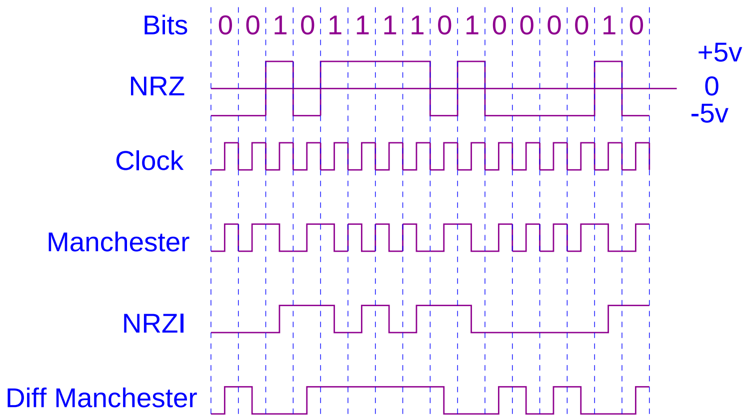
- Twisted Pair
  - Unshielded Twisted Pair (UTP)
    - Voice Grade: Telephone wire
    - Data Grade: Better quality

    $\Longrightarrow$ 100 Mbps over 50 m is possible
  - Shielded Twisted Pair (UTP)
- Coaxial Cable
- Optical Fiber
  - Modes:

$$\text{index of reflection} = \frac{\text{Speed in vacuum}}{\text{Speed in medium}}$$

  - Single mode
  - Multimode

Coding Design



- Non-Return to Zero (NRZ)
  - 1 = high level, 0 = low level
  - Problem: consecutive 1s or 0s ⟹ Unable to recover clock
    Uniform distribution of 1's and 0's tune the clocks
- Non-return to Zero Inverted (NRZI): Make a transition from the current signal to encode a one, and stay at the current signal to encode a zero; solves the problem of consecutive ones.
  - 0 = no transition at beginning of interval (one bit at time)
  - 1 = transition at beginning of interval

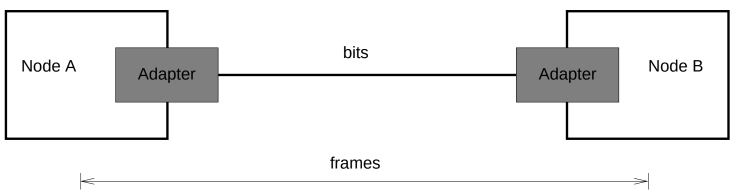
- Manchester:
  - 0 = low to high
  - 1 = high to low
- Differential Manchester
  - 1 = absence of transition
  - 0 = presence of transition

  Always a transition in middle of interval ⟹ easy to synchronize

Framing

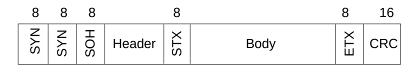
Overview

- Problem: Breaking sequence of bits into a frame
  - Must determine first and last bit of the frame
  - Typically implemented by network adaptor
  - Adaptor fetches (deposits) frames out of (into) host memory

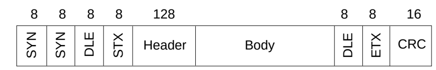


Byte-Oriented Protocols

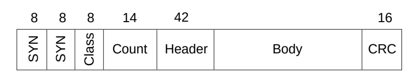
- Sentinel Approach
  - BISYNC(binary sync. comm.)



  - IMP-IMP (ARPANET)



  - Problem: ETX character might appear in the data portion of the frame.
  - Solution: Escape the ETX character with a DLE character in BISYNC; escape the DLE character with a DLE character in IMP-IMP.
- Byte Counting Approach (DDCMP)



  - Problem: Count field is corrupted (framing error).
  - Solution: Catch when CRC fails.

Bit-Oriented Protocols

- HDLC: High-Level Data Link Control (also SDLC and PPP)
- Delineate frame with a special bit-sequence: 01111110

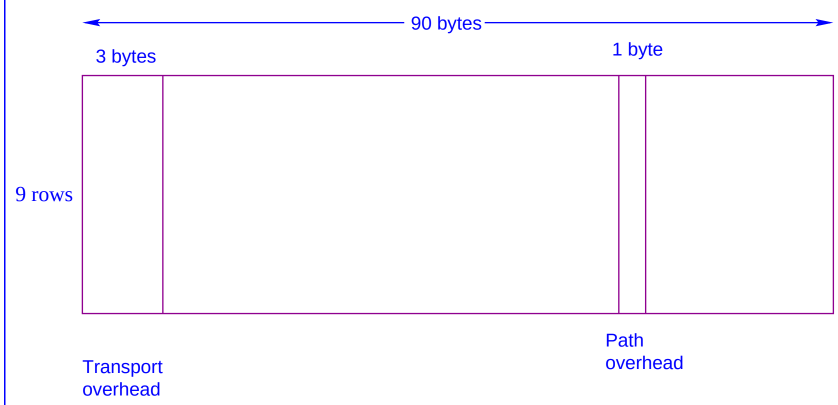
8	16		16	8
Beginning Sequence	Header	Body	CRC	Ending Sequence

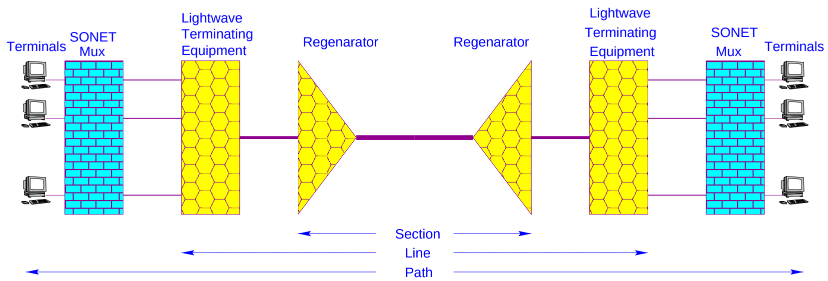
- Bit Stuffing
  - Sender: any time five consecutive 1s have been transmitted from the body of the message, insert a 0.
  - Receiver: should five consecutive 1s arrive, look at next bit(s):
    - if next bit is a 0: remove it
    - if next bits are 10: end-of-frame marker
    - if next bits are 11: error

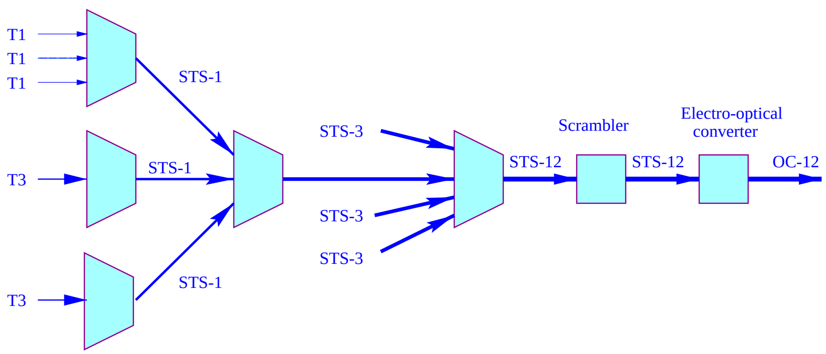
Clock-Based Framing

- SONET: Synchronous Optical Network
- ITU standard for transmission over fiber
- STS-1 (51.84 Mbps)
- Byte-interleaved multiplexing
- Each frame is 125μs long.

STS-1 Framce Structure







Error Detection

- Let P_b be the probability that a bit is in error
- Let F be the frame size in bits

$$\text{Probability[frame has no error]} = (1 - P_b)^F$$

$$\text{Probability[one or more bits in error]} = 1 - (1 - P_b)^F$$

- Example: Let F=1000 bits and $P_b = 10^{-6}$

$$\text{Pr[frame is in error]} = 1 - (1 - 10^{-6})^1 000 = 10^{-3}$$

Parity Checks

	0	1	2	3	4	5	6	7	
	1	0	1	1	0	1	1		Codeword
Odd parity	1	0	1	1	0	1	1	0	# of 1's is odd
Even parity	1	0	1	1	0	1	1	1	# of 1's is even

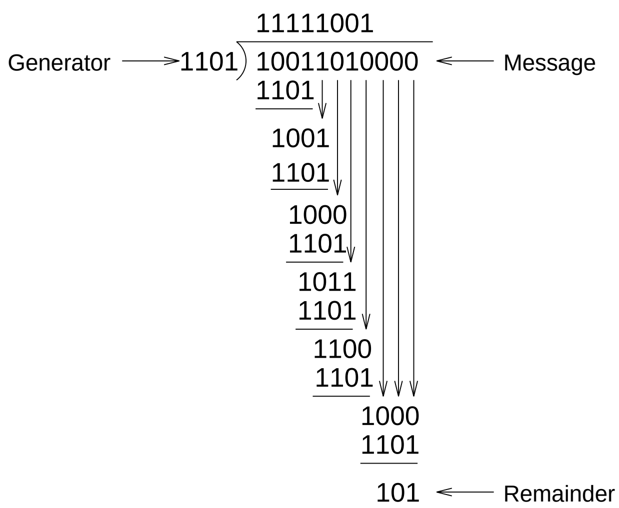
- Single error can be detected

Check Digit Method

- Make the number divisible by 9
- Example: 823 to be sent
  1. Left shift 823 $\Longrightarrow$ 8230
  2. Divide by 9 and find remainder $\Longrightarrow$ 4
  3. Subtract remainder from 9 $\Longrightarrow$ 9-4=5
  4. Add the result of step 4 to step 1:8235
  5. Check that the result is divisible by 9
- Detects all single-digit errors: 7235, 8335, 8255, 8237
- Detects several multiple-digit errors: 8765, 7346
- Does not detect some errors: 7335, 8775,
- Homework: Prove why it detects all single-digit errors

Cyclic Redundancy Check

- Add k bits of redundant data to an n-bit message.
- Represent n-bit message as an $n-1$ degree polynomial; e.g., MSG=10011010 corresponds to $M(x) = x^7 + x^4 + x^3 + x^1$.
- Let k be the degree of some divisor polynomial $C(x)$; e.g., $C(x) = x^3 + x^2 + 1$.
- Transmit polynomial $P(x)$ that is evenly divisible by $C(x)$, and receive polynomial $P(x) + E(x)$; $E(x)$=0 implies no errors.
- Recipient divides $(P(x) + E(x))$ by $C(x)$; the remainder will be zero in only two cases: $E(x)$ was zero (i.e. there was no error), or $E(x)$ is exactly divisible by $C(x)$.
  - Choose $C(x)$ to make second case extremely rare.
- Sender:
  - multiply $M(x)$ by x^k; for our example, we get $x^{10} + x^7 + x^6 + x^4$ (10011010000);
  - divide result by $C(x)$ (1101);
  - Send 10011010000 - 101 = 10011010101, since this must be exactly divisible by $C(x)$;
- Want to ensure that $C(x)$ does not divide evenly into polynomial $E(x)$.



- What can be detected?
  - All single-bit errors, as long as the x^k and x^0 terms have non-zero coefficients.
  - All double-bit errors, as long as $C(x)$ has a factor with at least three terms.
  - Any odd number of errors, as long as $C(x)$ contains the factor $(x + 1)$.
  - Any 'burst' error (i.e sequence of consecutive errored bits) for which the length of the burst is less than k bits.
  - Most burst errors of larger than k bits can also be detected.

Common polynomials for $C(x)$

CRC	$C(x)$
CRC-8	$x^8 + x^2 + x^1 + 1$
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^1 + 1$
CRC-12	$x^{12} + x^{11} + x^3 + x^2 + 1$
CRC-16	$x^{16} + x^{15} + x^2 + 1$
CRC-CCITT	$x^{16} + x^{12} + x^5 + 1$
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} +$ $x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

- Ethernet and FDDI use CRC-32

Two-Dimensional Parity

	0	1	2	3	4	5	6	7	
	0	1	0	1	0	0	1	1	← parity bit
	1	1	0	1	0	0	1	0	
	1	0	1	1	1	1	0	1	
	0	0	0	1	1	1	0	1	
	0	1	1	0	1	0	0	1	
	1	0	1	1	1	1	1	0	
Parity byte	1	1	1	1	0	1	1	0	

- 2D parity catches 1, 2, and 3-bit errors, and most 4-bit errors.
- Homework: Show this is true

Internet Checksum Algorithm

- The third approach
- View message as a sequence of 16-bit integers.
- Add these integers together using 16-bit ones complement arithmetic, and then take the ones complement of the result.
- That 16-bit number is the checksum.
- Unlike CRC, it doesn't have very strong error detection property
- The algorithm is easier to implement

Reliable Transmission

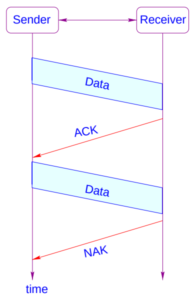
- Recover from corrupt frames
  - Error Correction Codes (ECC); also called Forward Error Correction (FEC)
  - Acknowledgments and Timeouts; also called Automatic Repeat request (ARQ)
- Delivers frames without errors, in proper order to network layer

Error Correction Mechanisms

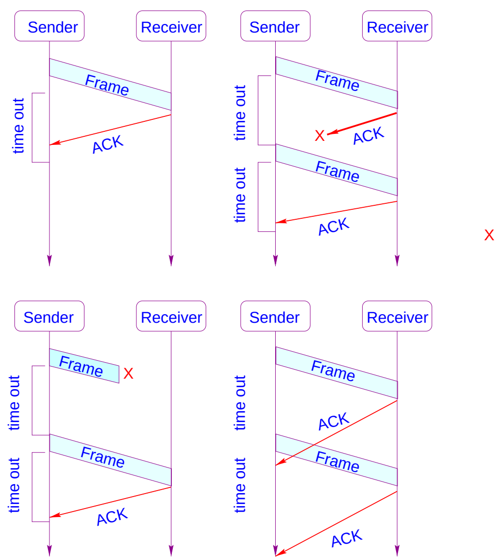
- ACK/NAK: provide sender some feed-back about the other end
- Time-out: for the case when entire packet or ACK is lost
- Sequence numbers: to distinguish retransmissions

Automatic Repeat Request (ARQ)

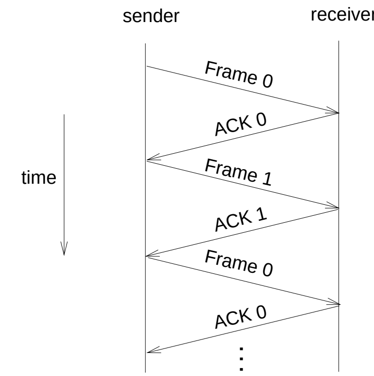
- Error detection
- Acknowledgment
- Retransmission after timeout
- Negative acknowledgment



ARQ Scenarios



Stop-and-Wait

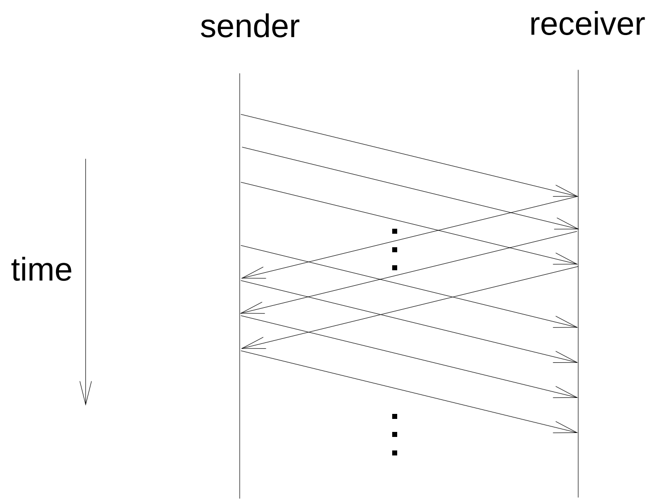


Problem: Keeping the pipe full.

Example: 1.5Mbps link $\times$ 45ms RTT $=$ 67.5Kb (8KB). Assuming frame size of 1KB, stop-and-wait uses about one-eighth of the link's capacity. Want the sender to be able to transmit up to 8 frames before having to wait for an ACK.

Sliding Window

Idea: Allow sender to transmit multiple frames before receiving an ACK, thereby keeping the pipe full. There is an upper limit on the number of outstanding (un-ACKed) frames allowed.



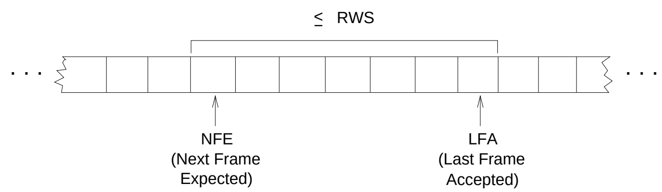
Sender:

- Assign sequence number to each frame (SeqNum)
- Maintain three state variables:
  - send window size (SWS)
  - last acknowledgment received (LAR)
  - last frame sent (LFS)
- Maintain invariant: LFS - LAR $\leq$ SWS

- When ACK arrives, advance LAR, thereby opening window
- Buffer up to SWS frames

Receiver:

- Maintain three state variables:
  - receive window size (RWS)
  - last frame acceptable (LFA)
  - next frame expected (NFE) or last frame received (LFR = NFE - 1)
- Maintain invariant: LFA - LFR $\leq$ RWS *or equivalently* LFA - NFE + 1 $\leq$ RWS



- Frame SeqNum arrives:
  - if NFE $\leq$ SeqNum $\leq$ LFA $\longrightarrow$ accept
  - if SeqNum $<$ NFE or SeqNum $>$ LFA $\longrightarrow$ discarded
- Send cumulative ACK
- Variations
  - selective acknowledgements
  - negative acknowledgements (NAK)

Sequence Number Space

- SeqNum field is finite; sequence numbers wrap around
- Sequence number space must be larger than number of outstanding frames
- SWS $\leq$ MaxSeqNum-1 is not sufficient
  - suppose 3-bit SeqNum field (0..7)
  - SWS=RWS=7
  - sender transmit frames 0..6
  - arrive successfully, but ACKs lost
  - sender retransmits 0..6
  - receiver expecting 7,0..5, but receives second incarnation of 0..5
- SWS $<$ (MaxSeqNum+1)/2 is correct rule
- Intuitively, SeqNum "slides" between two halves of sequence number space

Concurrent Logical Channels

- Multiplex several logical channels over a single point-to-point link; run stop-and-wait on each logical channel.
- Maintain three bits of state for each channel:
  - boolean saying whether the channel is currently busy
  - sequence number for frames sent on this logical channel
  - next sequence number to expect on this logical channel
- ARPANET supported eight logical channels over each ground link (16 over each satellite link).
- Header for each frame included a 3-bit channel number and a 1-bit sequence number, for a total of 4 bits; same number of bits as the sliding window protocol requires to support up to eight outstanding frames on the link.
- Separates reliability from *flow control* and *frame order*.

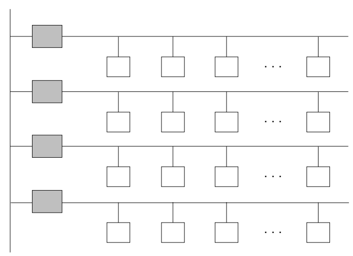
Ethernet

Overview

- History
  - Developed by Xerox PARC in mid-1970s
  - Roots in Aloha packet-radio network
  - Standardized by Xerox, DEC, and Intel in 1978
  - Similar to IEEE 802.3 standard
- CSMA/CD
  - carrier sense
  - multiple access
  - collision detection
- Bandwidth: 10Mbps and 100Mbps
- Problem: Distributed algorithm that provides fair access to a shared medium

Physical Properties

- Classical Ethernet (thick-net)
  - maximum segment of 500m
  - transceiver taps at least 2.5m apart
  - connect multiple segments with repeaters
  - no more than 2 repeaters between any pair of nodes (1500m total)
  - maximum of 1024 hosts
  - also called 10Base5



- Alternative technologies
  - 10Base2 (thin-net): 200m; daisy-chain configuration
  - 10BaseT (twisted-pair): 100m; star configuration

Frame Format

64	48	48	16		32	8
Preamble	Dest Addr	Src Addr	Type	Body	CRC	Postamble

Addresses:

- Unique, 48-bit unicast address assigned to each adaptor
- Example: `8:0:2b:e4:b1:2`
- Broadcast: all 1s
- Multicast: first bit is 1

Adaptor receives all frames; it accepts (passes to host):

- Frames addressed to its own unicast address
- Frames addressed to the broadcast address
- Frames addressed to any multicast address it has been programmed to accept
- All frames when in promiscuous mode

Transmitter Algorithm

If line is idle:

- Send immediately
- Upper bound message size of 1500 bytes
- Must wait 51μs between back-to-back frames

If line is busy:

- Wait until idle and transmit immediately
- Called *1-persistent* (special case of *p-persistent*)

If collision:

- jam for 512 bits, then stop transmitting frame
- minimum frame is 64 bytes (header + 46 bytes of data)
- delay and try again
  - 1st time: uniformly distributed between 0 and 51.2μs
  - 2nd time: uniformly distributed between 0 and 102.4μs
  - 3rd time: uniformly distributed between 0 and 204.8μs
  - give up after several tries (usually 16)
  - exponential backoff

Experiences

Observe in Practice

- 10-200 hosts (not 1024)
- Length shorter than 1500m (RTT closer to 5μ than 51μ)
- Packet length is bimodal
- High-level flow control and host performance limit load

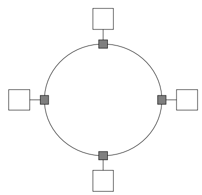
Recommendations

- Do not overload (30% utilization is about max)
- Implement controllers correctly
- Use large packets
- Get the rest of the system right (broadcast, retransmission)

FDDI

Overview

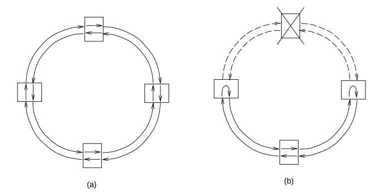
- Token Ring Networks
  - PRONET: 10Mbps and 80 Mbps rings
  - IBM: 4Mbps token ring
  - 16Mbps IEEE 802.5/token ring
  - 100Mbps Fiber Distributed Data Interface (FDDI)



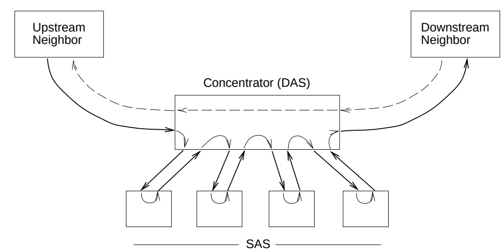
- Basic Idea
  - frames flow in one direction: upstream to downstream
  - special bit pattern (token) rotates around ring
  - must capture token before transmitting
  - release token after done transmitting
    - immediate release
    - delayed release
  - remove your frame when it comes back around
  - stations get round-robin service

Physical Properties of FDDI

Dual Ring Configuration



Single and Dual Attachment Stations



- Each station imposes a delay (e.g., 50ns)
- Maximum of 500 stations
- Upper limit of 100km (200km of fiber)
- Uses 4B/5B encoding
- Can be implemented over copper (CDDI)

Timed Token Algorithm

- Token Holding Time (THT): upper limit on how long a station can hold the token.
- Token Rotation Time (TRT): how long it takes the token to traverse the ring.

  $\text{TRT} \leq \text{ActiveNodes} \times \text{THT} + \text{RingLatency}$
- Target Token Rotation Time (TTRT): agreed-upon upper bound on TRT.
- Algorithm
  - each node measures TRT between successive arrivals of the token
  - if measured TRT $>$ TTRT, then token is late so don't send data
  - if measured TRT $<$ TTRT, then token is early so OK to send data
  - define two classes of traffic
    - synchronous data: can always send
    - asynchronous data: can send only if token is early
  - worse case: $2\times$TTRT between seeing token
  - not possible to have back-to-back rotations that take $2\times$TTRT time

Token Maintenance

- Lost Token
  - no token when initializing ring
  - bit error corrupts token pattern
  - node holding token crashes
- Generating a Token (and agreeing on TTRT)
  - execute when join ring or suspect a failure
  - each node sends a special *claim frame* that includes the node's *bid* for the TTRT
  - when receive claim frame, update bid and forward
  - if your claim frame makes it all the way around the ring:
    - your bid was the lowest
    - everyone knows TTRT
    - you insert new token
- Monitoring for a Valid Token
  - should see valid transmission (frame or token) periodically
  - maximum gap $=$ ring latency $+$ max frame $\leq$ 2.5ms
  - set timer at 2.5ms and send claim frame if it fires

Frame Format

Start of Frame	Control	Dest Addr	Src Addr	Body	CRC	End of Frame	Status
8	8	48	48		32	8	24

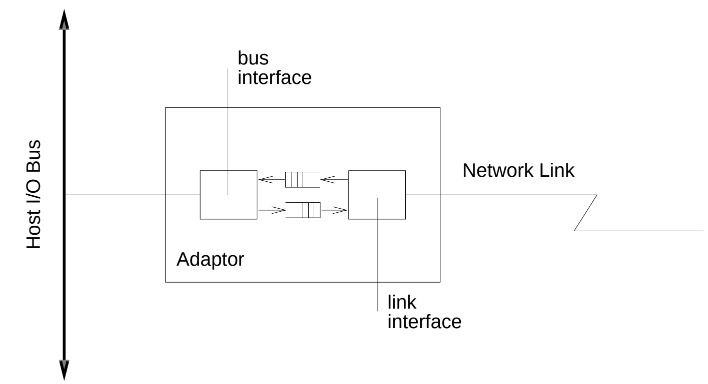
- Control Field
  - 1st bit: asynchronous (0) versus synchronous (1) data
  - 2nd bit: 16-bit (0) versus 48-bit (1) addresses
  - last 6 bits: demux key (includes reserved patterns for token and claim frame)
- Status Field
  - from receiver back to sender
  - error in frame
  - recognized address
  - accepted frame (flow control)

Network Adaptors

Overview

Typically where data link functionality is implemented

- Framing
- Error Detection
- Media Access Control (MAC)



Host Perspective

Control Status Register (CSR)

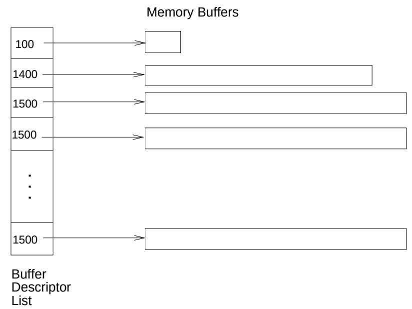
- Available at some memory address
- CPU can read and write
- CPU instructs Adaptor (e.g., transmit)
- Adaptor informs CPU (e.g., receive error)

Example

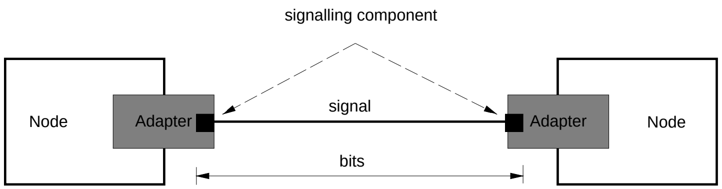
```
LE_RINT  0x0400  Received packet Interrupt (RC)
LE_TINT  0x0200  Transmitted packet Interrupt (RC)
LE_IDON  0x0100  Initialization Done (RC)
LE_IENA  0x0040  Interrupt Enable (RW)
LE_INIT  0x0001  Initialize (RW1)
```


Moving Frames Between Host and Adaptor

Direct Memory Access (DMA)



Programmed I/O (PIO)



Device Driver

Interrupt Handler

```
interrupt_handler()
{
    disable_interrupts();
    /* some error occurred */
    if (csr & LE_ERR)
    {
        print_and_clear_error();
    }
    /* transmit interrupt */
    if (csr & LE_TINT)
    {
        csr = LE_TINT | LE_INEA;
        semSignal(xmit_queue);
    }
    /* receive interrupt */
    if (csr & LE_RINT)
    {
        receive_interrupt();
    }
    enable_interrupts();
    return(0);
}
```


Transmit Routine:

```
transmit(Msg *msg)
{
    char *src, *dst;
    Context c;
    int len;

    semWait(xmit_queue);
    semWait(mutex);
    disable_interrupts();
    dst = next_xmit_buf();
    msgWalkInit(&c, msg);
    while ((src = msgWalk(&c, &len)) != 0)
        copy_data_to_lance(src, dst, len);
    msgWalkDone(&c);
    enable_interrupts();
    semSignal(mutex);
    return;
}
```

Receive Interrupt Routine

```
receive_interrupt()
{
   Msg  *msg,  *new_msg;
   char *buf;

   while (rdl = next_rcv_desc())
   {
       /* create process to handle this message */
       msg = rdl->msg;
       process_create(ethDemux, msg);

       /* msg eventually freed in ethDemux */
       /* now allocate a replacement */
       buf = msgConstructAllocate(new_msg, MTU);
       rdl->msg = new_msg;
       rdl->buf = buf;
       install_rcv_desc(rdl);
   }
   return;
}
```