DRIFT: Efficient Message Ordering in Ad Hoc Networks Using Virtual Flooding

Stefan Pleisch[1] Thomas Clouser[2] Mikhail Nesterenko[2*] André Schiper[1]

[1]Distributed Systems Laboratory (LSR)
Swiss Federal Institute of Technology (EPFL), CH-1015 Lausanne, Switzerland

[2]The Department of Computer Science
Kent State University, Kent, OH 44242, USA

stefan.pleisch@epfl.ch tclouser@kent.edu mikhail@cs.kent.edu andre.schiper@epfl.ch

Abstract

We present DRIFT — a total order multicast algorithm for ad hoc networks with mobile or static nodes. Due to the ad hoc nature of the network, DRIFT uses flooding for message propagation. The key idea of DRIFT is virtual flooding *— a way of using unrelated message streams to propagate message causality information in order to accelerate message delivery. We describe DRIFT in detail. We evaluate its performance in a simulator and in a wireless sensor network. In both cases our results demonstrate that the performance of DRIFT exceeds that of the simple total order multicast algorithm designed for wired networks, on which it is based. In simulation at scale, for certain experiment settings, DRIFT achieved speedup of several orders of magnitude.*

1 Introduction

Recent advances in PDA and wireless networked sensor technology enable the ad hoc networks of these devices to handle increasingly sophisticated tasks. As the reliance on these devices grows, so does the need to bring well-established communication primitives to such networks. One such primitive is total order multicast. As a motivating example, consider that a temporary military sensor network is deployed to protect an extended valuable asset. The sensor network does not have any routing infrastructure: the communication is multi-hop and ad hoc. Several operators move through the field and periodically issue directives for all sensor nodes to change the mode of surveillance or focus on particular targets of observation. It is mission-critical that the directives are delivered in the same order at each sensor node. Otherwise, different parts of the network may start performing conflicting tasks. Thus, the directives need to be sent using total order multicast.

Total order multicast has been studied extensively, predominantly in wired networks. An order is imposed on the multicast messages and all nodes are expected to deliver them in this order. One ordering approach is to arrange messages according to causal precedence. Concurrent messages are arranged in some deterministic order, e.g., according to the sender's identifier. The nodes buffer the received messages and then *deliver* them to the application in this order. Traditionally, total order multicast algorithms do not consider the routing aspect of message transmission and assume that the network is completely connected (each node participating in the multicast has a channel to every other node). However, maintaining such routing infrastructure may not be feasible in ad hoc networks, especially if nodes are mobile, as in the above scenario. Thus, due to node mobility and large scale of the network either proactive or reactive route maintenance may not be efficient. Hence, traditional total order multicast algorithms may not be applicable to such networks.

In such networks, *flooding* is an effective mechanism of reaching all nodes in the network without underlying routing infrastructure. In its simple form, a flooding source broadcasts a message to its neighbors and all other nodes rebroadcast the flooded message exactly once. Note that we distinguish between a network-wide flooding and a *(local) radio broadcast*, which

*This research was supported in part by NSF CAREER Award 0347485.

is a transmission that is received by all nodes within transmission range of the broadcasting node. The use of flooding requires nodes to forward messages sent to other nodes. Thus, there is an opportunity to piggyback information on the rebroadcast messages. We call this technique *virtual flooding*. We apply it to a total order multicast algorithm inspired by Lamport's algorithm [19]. The resulting total order multicast algorithm, which we call DRIFT, is optimized for ad hoc networks and enables the recipients to deliver the received messages faster. We present simulation and implementation results that demonstrate significant performance gains due to virtual flooding.

The remainder of the paper is structured as follows. In Section 2 we introduce the total order multicast problem and survey existing work. In Section 3 we present virtual flooding. We give a detailed description of DRIFT in Section 4. In Section 5 we present the simulation results of DRIFT's operation. In Section 6 we describe the implementation of DRIFT in a wireless sensor network and present the obtained results. We address the extensions of DRIFT to accomodate failures and changes in network membership in Section 7. We conclude the paper in Section 8.

2 Total Order Multicast and Ad Hoc Networks

Total order multicast (or TO-multicast)[1] is a fundamental communication mechanism utilized by a variety of applications. It has two communication primitives: TO-multicast and TO-deliver. An application program invokes *TO-multicast* to send a message to all the nodes of the multicast group. To ensure that the recipients agree on the delivery order, they may buffer and reorder the received messages. Once the message order is established, a node executes *TO-deliver* to convey the message to the application.

2.1 Ad Hoc Network Specifics

The network consists of a set of radio-communication capable nodes. A subset of these nodes are *sources* — Σ and may invoke TO-multicast, while another subset are *destinations* — Δ invoking TO-deliver. The two sets, in general, are not related as a source may not have to TO-deliver messages. Some nodes in the network may be in neither set: they act only as message forwarders.

Certain properties of ad hoc networks differentiate them from conventional wired networks. Communication between two nodes is immediate if the two nodes are within transmission range of each other. Otherwise, intermediate nodes may have to forward the message along multiple hops from the source to the destination. The nodes may potentially be mobile which further complicates communication. Network and individual node resources such as available bandwidth, battery power, memory size, etc. may be limited.

In such setting, it may not be feasible to maintain routing infrastructure. Instead, message flooding may be used as a predominant communication primitive. Hence the need to develop a TO-multicast algorithm specifically optimized to use flooding. Before we describe the algorithm, we survey TO-multicast algorithms already published in the literature.

2.2 TO-Multicast Algorithms Overview

TO-multicast algorithms typically assume the existence of a reliable message delivery mechanism which guarantees that all nodes receive the multicast message. A variety of TO-multicast algorithms are described in the literature. Défago et al in their survey paper [12] classify the algorithms according their ordering techniques: *sequencer-based*, *privilege-based*, *destination*, and *communication history*. For brevity, our overview of TO-multicasts in wired networks is deliberately incomplete: we cite one or two typical examples per technique. For detailed discussion and comparison of TO-multicast algorithms we refer the reader to the original paper [12]. Few TO-multicast algorithms have been proposed for ad hoc networks. This overview motivates communication history ordering as a TO-multicast technique of choice for DRIFT.

Sequencer-based ordering. In this approach one node is selected as the *sequencer*. Every node that wishes to TO-multicast a message contacts the sequencer and obtains a sequence number which is then used to determine the delivery order. To balance the load, the sequencer function can be successively performed by multiple nodes. An example of this approach for fixed networks is described by Navaratnam et al. [22]. Anastati et al. [4] and Bartoli [6] describe a sequencer-based TO-multicast for single-hop mobile networks. They consider an infrastructure-based network where a set of wired gateways order the multicast messages and ensure their transmission to the mobile nodes. In contrast, we do not make use of a stationary wired infrastructure in our algorithm. Moreover, wireless communication in our setting is multihop rather than single hop.

While sequencer-based algorithms may perform well in fixed networks, they may not be applicable to ad hoc networks. In particular, the sequencer and a routing path to it needs to be known to all the sources. The necessity of a single sequencer limits the scalability of this approach. Notice also that before a message is TO-multicast to the destinations, an additional point-to-point message communication from the source to the sequencer is usually required. In an ad hoc network this may increase message delivery latency and add message overhead.

[1] Total order multicast is sometimes also called *atomic multicast*.

Privilege-based ordering. In this type of algorithms, the source TO-multicasts a message when the source is granted an exclusive privilege to do so. One way to ensure exclusivity is to circulate a single token among sources. A source can TO-multicast a message only when it holds the token. An example of such algorithms in wired networks is Train [11]. A token-based algorithm in mobile ad hoc networks is described by Malpani et al. [21]. Token-based algorithms require maintenance of routing information. They also require token maintenance and recovery. Thus, such algorithms may not always be practical in ad hoc networks.

Destination ordering. In this approach, the destinations (possibly an *agreement subset* of these nodes) agree on the message delivery order. An example of this class is the TO-multicast algorithm by Chandra and Toueg [10]. This approach requires extensive communication within the agreement set and between this set and the other destinations. Thus, destination ordering may not be appropriate for ad hoc networks.

Communication history ordering. The algorithms of this class deliver messages based on the causal order of multicasts. Causal relation [19] establishes a partial order of messages. This partial order is expanded to total order by delivering concurrent messages in some deterministic way. There is a number of communication history-based algorithms for wired networks [8, 19, 25]. Prakash et al. [24] describe a communication history-based TO-multicast algorithm for mobile networks. Unlike DRIFT, their algorithm uses wired infrastructure. Communication history-based ordering is rather promising for ad hoc networks as it is entirely distributed and it scales well as there is no need for extra ordering messages. DRIFT belongs to this class.

Probabilistic multicast. Luo et al. [20] explore a probabilistic approach to total order multicast in ad hoc networks. Their algorithm guarantees delivery with a certain probability. In contrast, in this paper we focus on TO-multicast with deterministic guarantees.

2.3 The Problem of Communication History Ordering in Ad Hoc Networks

As we discussed the advantages of communication-history ordering approach to TO-multicasting for ad hoc networks, we shall now focus on the specifics of this type of design by presenting Lamport's algorithm [19, 12] (which is the basis of DRIFT). This algorithm assumes FIFO communication channels and reliable message delivery. It is based on logical clocks. Before TO-multicasting a message, the source increments its logical clock and timestamps the message with this new clock value. Each destination TO-delivers the messages in the increasing order of timestamps. Messages with identical clock values (these messages have been sent concurrently) are delivered in some deterministic order, e.g., in the order of their senders' identifiers. Since message receipt is reliable, every node TO-delivers the messages in the same order.

The main difficulty in Lamport's approach is to delay the message delivery long enough to ensure that messages with smaller timestamps are not received in the future. This is handled as follows. Note that every source monotonically increases the timestamps it assigns to the multicast messages. Since messages from the same source are received in FIFO order, once a destination receives a message with a certain timestamp, all successive messages from this source will bear greater timestamps. Every destination n stores the latest received timestamp for each source. The messages are delivered according to the following rule. Node n can TO-deliver a particular message m only after it receives a message with a higher timestamp from every source. Due to the FIFO message delivery, this guarantees that in the future n will not receive messages with timestamps smaller than that of m.

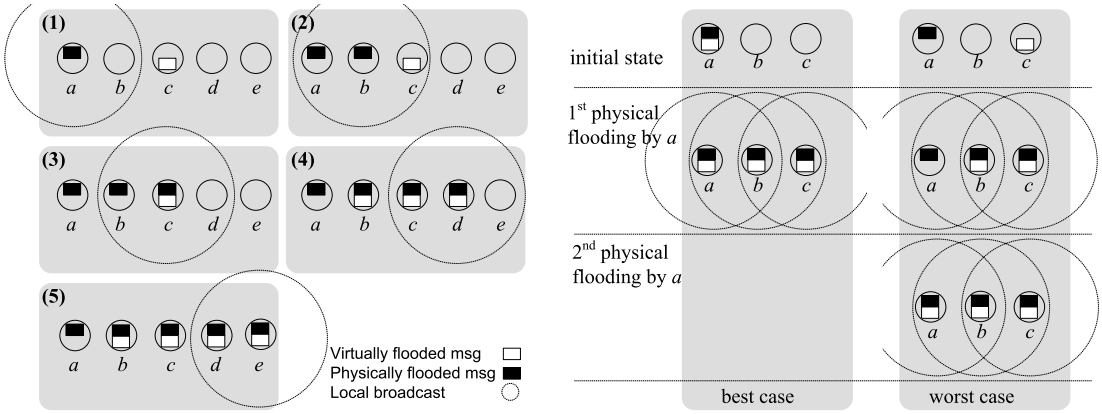
Hence, the delivery rate of all destinations depends on the sending rate of the source that multicasts least frequently. Moreover, as described, Lamport's algorithm is not terminating: to ensure delivery at all destinations, each source has to continuously multicast messages. The delivery can be implemented by requiring that each node periodically multicasts a dummy message. The only purpose for such dummy message is to notify the other destinations of the source's most recent logical clock value. However, as this approach introduces extra message overhead it may be impractical. We propose an alternative technique to propagate recent logical clock values of the sources. Our approach exploits the properties of ad hoc networks. We call this technique virtual flooding.

3 Virtual Flooding

Virtual flooding distributes data to every node in the network by attaching it to unrelated messages propagated in the network. Virtual flooding is different from *physical flooding* (or just *flooding*) as it does not require any extra messages to be sent. Specifically, to propagate virtually flooded data, a node attaches the data to physically flooded message it has to locally broadcast. Consider the example in Fig. 1(i). The network consists of five nodes a through e in a line. The message transmission range for each node only covers its immediate neighbors. Node a *physically* floods message m (represented by a black box in the figure). Node c *virtually* floods message m' (white box). When m reaches c (see Fig. 1(i 3)), c attaches m' to m and

resends $m||m'$. Nodes b and d receive the joint message (see (i 4)). Node d resends the joint message again. Thus, with a single physical flood, the virtually flooded message m' reaches all nodes in the network except a. Another physical flood from any node in the network results in a receiving m' (see (i 5)).

The number of physical floods required to propagate a virtually flooded message varies. In the worst case this number is proportional to the diameter of the network. Consider the example in Fig 1(ii). In the best case node a contains messages for both virtual and physical flooding. In this case only one physical flood is required. However, in case the virtually and physically flooded messages are located at the opposite ends of the network, it takes two floods to propagate them.



(i) Virtual flooding example. (ii) Efficiency of virtual flooding.

Figure 1. Virtual flooding example.

Provided that sufficiently many physical floods occur, virtually flooded messages eventually reach all nodes in the network. While it increases the size of physically flooded messages, it results in better bandwidth utilization as the virtually flooded data does not require separate messages. Thus, there is no overhead incurred in acquiring the radio channel and no extra message headers are required. This advantage is particularly important if the virtually flooded data is relatively small in size like the causality information virtually flooded by DRIFT as we describe in the next section.

4 DRIFT Description

The key idea of DRIFT is to use virtual flooding to propagate information about the last logical clock values of the other sources seen by some source. This approach lowers delivery latency. In this section, we describe how virtual flooding is utilized in DRIFT. We then describe the algorithm, and demonstrate its operation with an example. We conclude the section with the discussion of how DRIFT is to be efficiently implemented in practical ad hoc networks.

Initially, we assume that destinations are static. Each flooded message is reliably received by every node. Multiple messages from an individual source are received by each node in FIFO order. Nodes do not crash. The sources do not join or leave the network (i.e. we consider *static* group membership). Furthermore, we assume that at least one source sends an infinite number of messages. Later we discuss how these assumptions may be relaxed or implemented.

4.1 Virtual Flooding in DRIFT

DRIFT extends Lamport's TO-multicast. It uses virtual flooding to propagate timestamp information and alleviate the need for periodic dummy message transmission. The idea is as follows. Suppose node p receives message m from source q with timestamp ts. Observe that to safely deliver m, p does not necessarily need to receive a message with timestamp $ts' > ts$ from another source r. It is sufficient that p learns that it will not receive a message from r with a timestamp less than or equal to ts. When a source selects a new timestamp for the message to multicast, the timestamp is chosen such that it exceeds the timestamps of the messages that the source has received. Thus, if p learns that r received a message with a timestamp ts or greater, it can safely deliver m. In DRIFT, each source virtually floods its current logical clock value.

Recall that as presented in Section 3, all virtually flooded data reaches every node. Yet, in our case, only the freshest logical clock values are of significance. Hence, in DRIFT, this information is updated at every node and only the most recent logical clock information per source is resend with each physical message. This causes the virtually flooded information to be constantly updated along the way.

Although we assume that the messages multicast by a single source are received by each node in FIFO order, the virtual flooding information is attached to arbitrary messages. Thus, the timestamps carried by virtual flooding may overtake the ones

carried by physical messages. For example, suppose node p multicasts a message with timestamp t_1 and later virtually floods $t_2 > t_1$. It may happen that some node q receives a message carrying t_2 in its virtual flooding part earlier than the message with t_1. If q uses t_2 to deliver some message with timestamp t_3 such that $t_1 < t_3 < t_2$ the total order is violated. Thus, care must be taken when delivering a message based on timestamp information received via virtual flooding. In DRIFT we use sequence numbers to relate physically and virtually flooded timestamps.

4.2 Algorithm Description

The pseudocode of DRIFT for each node p is shown in Fig. 2. Every source ($p \in \Sigma$) maintains its logical clock lc as well as sequence number sn of the last message that it multicasts. Every node maintains a set of received message information as well as a set *Seen* to keep track of virtual flooding information. Each destination ($p \in \Delta$) also maintains the sets of ready for delivery — *READY* and delivered — *DLVD* message information. In addition, each destination has an array *RcvdSN* to store the last sequence number of a message received from each respective source. DRIFT contains two actions. The first action — **TO-multicast**(m) is invoked when the application requires to multicast a message m. The second action — message receipt, is executed when p receives a message. Function **getHighestTimestamp** is used as a shorthand for repeated operation of selecting highest-timestamped entries out of *Seen* in both actions.

If a source p has a message m to multicast, it executes **TO-multicast**. By executing this action p obtains a new timestamp (lc) and a new sequence number for the message. This information is entered in *Seen*. Node p then broadcasts the message to its neighbors. The freshest virtual flooding information is attached to the message. Specifically, **TO-multicast** invokes **getHighestTimestamp** which selects from *Seen* the highest timestamped entry for each source.

When p receives a message, it performs the following three operations (see Fig. 2): virtual flooding update (*vf update*), received message processing (*rcpt processing*), and message delivery (*delivery*). Notice that sources that are at the same time also destinations process their own messages similar to the messages received from other sources. In virtual flooding update p merges its own virtual flooding data in *Seen* with that carried by the received message *qSeen*. In the second operation p checks if the received message is new. If so, p adds the message information to *RCVD*. If p is a source, it updates it local clock and virtual flooding information about itself in *Seen*. If p is a destination, it updates the sequence number of the last received message from the source in *RcvdSN*. Then p rebroadcasts the message. Note that the message is forwarded with the most up-to-date virtual flooding data. In case p is a destination, after received message processing, p evaluates if any of the buffered messages are ready for delivery. The procedure is as follows. Destination p forms a set of candidates for delivery *READY*. A candidate $\langle um, u, usn, uts\rangle$ is an undelivered message with the following characteristics: for each source i there is an entry $\langle i, isn, its\rangle$ in virtual flooding set *Seen* such that this entry corresponds to a message already received by p: $RcvdSN[i] = is$ and the timestamp of the candidate message is less than the timestamp of the source $uts < its$; or, in case the timestamps are equal ($uts = its$), the source identifiers are used to break a tie ($u \leq i$). After forming the candidate set *READY*, p repeatedly examines the set and select the message with the smallest timestamp. Again, the source identifiers are used to break a tie. The selected message is TO-delivered.

4.3 Example operation.

We demonstrate the operation of DRIFT with an example (see Fig. 3). The example network has four nodes: $\{a, b, c, d\}$ out of which two — a and b are sources and the other two are destinations. Node a multicasts messages m_1 and m_2, while b multicasts m_3. In our example we focus on the delivery of the messages at destinations c and d and skip unrelated events.

4.4 Implementation Considerations

Optimizing data structures. Some of the wireless ad hoc platforms have limited memory resources (e.g. Crossbow's MICA2 motes [15]). The data structures used in DRIFT can be optimized to reduce memory at each individual node. We now discuss some of these optimizations. Observe that there is no need to keep track of messages after they are TO-delivered. Thus, the function of sets *RCVD* and *DLVD* can be modified. Set *DLVD* can be disposed of altogether. Set *RCVD* can only keep the messages that are not yet delivered. With this modification, the candidate message selection proceeds as before. However, in the original version of DRIFT, *RCVD* is used to recognize duplicate messages in *rcpt processing* operation. Yet, since we assume single source FIFO message delivery, array *RcvdSN* can be used for this purpose. Specifically, if a node receives a message *qm* from source q with sequence number *qsn* and $RcvdSN[q] = qsn$ then the newly received message is a duplicate and should be discarded.

Set *Seen* can also be optimized. Notice that *Seen* only needs to contain the elements pertaining to undelivered messages. Once the message is delivered, all virtual flooding information about it can be removed. Moreover, according to the way the entries in *Seen* are used, for each node and each sequence number it is sufficient to store only the entry with the highest timestamp.

The size of *Seen* can be further decreased at the expense of message delivery latency. The modification is as follows. Set *Seen* keeps at most two entries per each source q. One entry has the highest timestamp for the sequence number of the last

node p

variables

if $p \in \Sigma$ — p is a source

lc — local logical clock, initially 0

sn — sequence number of last message multicast, initially 0

RCVD — received message info, initially $\varnothing$

Seen — virtual flooding info set, initially $\varnothing$

if $p \in \Delta$ — p is a destination

READY, *DLVD* — ready for delivery and delivered messages, initially $\varnothing$

RcvdSN — sequence number of the last received message for each i, initially all 0-s

actions

TO-multicast(m)

$lc := lc + 1$

$sn := sn + 1$

$Seen := Seen \cup \{\langle p, sn, lc\rangle\}$

broadcast$(m, p, sn, lc,$ **getHighestTimestamp**$(Seen))$

when receive$(qm, q, qsn, qts, qSeen)$

vf update: $Seen := Seen \cup qSeen$

rcpt processing: **if** $\langle qm, q, qsn, qts\rangle \notin RCVD$ **then** /* received for the first time */

$RCVD := RCVD \cup \{\langle qm, q, qsn, qts\rangle\}$

if $p \in \Sigma$ **then**

$lc :=$ **max**$(lc, qts) + 1$

$Seen := Seen \cup \{\langle p, sn, lc\rangle\}$

if $p \in \Delta$ **then**

$RcvdSN[q] := qsn$

broadcast$(qm, q, qsn, qts,$ **getHighestTimestamp**$(Seen))$

delivery: **if** $p \in \Delta$ **then**

$READY := \{\langle um, u, usn, uts\rangle \in RCVD \setminus DLVD \mid$
$\forall i \in \Sigma, \exists \langle i, isn, its\rangle \in Seen :$
$RcvdSN[i] = isn \wedge uts \leq its\}$

$DLVD := DLVD \cup READY$

while $READY \neq \varnothing$ **do**

let $\langle vm, v, vsn, vts\rangle \in READY$ be such that
$\forall \langle um, u, usn, uts\rangle \in READY : vts < uts \vee (vts = uts \wedge v \leq u)$

TO-deliver *vm*

$READY := READY \setminus \{\langle vm, v, vsn, vts\rangle\}$

function getHighestTimestamp$(Seen)$

$highestSeen = \varnothing$

foreach $i \in \Sigma$ **do**

let $\langle i, isn, its\rangle \in Seen$ be such that $\forall \langle i, isn', its'\rangle \in Seen : its' \leq its$

$highestSeen := highestSeen \cup \{\langle i, isn, its\rangle\}$

return$(highestSeen)$

Figure 2. DRIFT pseudocode

a sends: $\langle m_1, a, 1, 1, \{\langle a, 1, 1\rangle\}\rangle$
b sends: $\langle m_2, b, 1, 1, \{\langle b, 1, 1\rangle\}\rangle$
a sends: $\langle m_3, a, 2, 2, \{\langle a, 2, 2\rangle\}\rangle$
a forwards: $\langle m_2, b, 1, 1, \{\langle a, 2, 3\rangle\langle b, 1, 1\rangle\}\rangle$
b forwards: $\langle m_3, a, 2, 2, \{\langle a, 2, 2\rangle\langle b, 1, 2\rangle\}\rangle$
c receives m_1 : *RcvdSN* $= [1, 0]$, *Seen* $= \{\langle a, 1, 1\rangle\}$
cannot deliver m_1 since *Seen* does not have an entry for b
c receives m_2 via a : *RcvdSN* $= [1, 1]$, *Seen* $= \{\langle a, 1, 1\rangle, \langle a, 2, 3\rangle, \langle b, 1, 1\rangle\}$
delivers m_1 since its timestamp is $mts = 1$ and *Seen* has an entry for each source that
allows addition of m_1 to *READY*; specifically $\langle a, asn = 1, ats = 1\rangle \in$ *Seen*,
for this entry *RcvdSN*$[a] =$ *asn*,$mts = ats$ and $a \leq a$, notice that $\langle a, 2, 3\rangle \in$ *Seen* cannot be used
since the message with sequence number 2 is not received yet,
$\langle b, bsn = 1, bts = 1\rangle \in$ *Seen*, for this entry *RcvdSN*$[b] =$ *bsn*, $mts = bts$ and $a < b$
c receives m_3 via b : *RcvdSN* $= [2, 1]$, *Seen* $= \{\langle a, 1, 1\rangle, \langle a, 2, 2\rangle, \langle a, 2, 3\rangle, \langle b, 1, 1\rangle, \langle b, 1, 2\rangle\}$
forwards: $\langle m3, a, 2, 2, \{\langle a, 2, 3\rangle\langle b, 1, 2\rangle\}\rangle$ note updated entry for a in *qSeen*,
delivers m_2 and m_3
d receives m_2 : *RcvdSN* $= [0, 1]$, *Seen* $= \{\langle b, 1, 1\rangle\}$ cannot deliver messages
d receives m_1 : *RcvdSN* $= [1, 1]$, *Seen* $= \{\langle a, 1, 1\rangle, \langle b, 1, 1\rangle\}$ delivers m_1
d receives m_3 via b and c : *RcvdSN* $= [2, 1]$, *Seen* $= \{\langle a, 1, 1\rangle, \langle a, 2, 3\rangle\langle b, 1, 1\rangle, \langle b, 1, 2\rangle\}$
delivers m_2 and m_3

Figure 3. DRIFT: example operation. The depicted events happen in sequence. The sequence is from top to bottom.

received message *RcvdSN*$[q]$. This is the entry that is used in case the node gets virtual flooding data that there is an outstanding message form q. The other entry in *Seen* has the highest timestamp seen (either through message receipt or virtual flooding) from q. This entry is used if there are no outstanding messages. Notice that there is a potential delivery delay if there are multiple outstanding messages from the same source. Suppose messages m_1 and m_2 from q are in transit and are not received by node p. The messages' sequence numbers are 1 and 2 respectively. Node p learns through virtual flooding, that q had a timestamp ts_1 and sequence number 1. Later, p also learns that q had timestamp ts_2 and sequence number 2. Due to the limitations that are imposed on modified *Seen*, $\langle q, 2, ts_2\rangle$ has to replace $\langle q, 1, ts_1\rangle$. However, when p receives m_1, p cannot use ts_2 if messages are eligible for delivery as m_2 is still in transit and p no longer has access to ts_1. Notice that set *READY* is not necessary for implementation. Each node p can just maintain *RCVD* sorted in timestamp order. For delivery evaluation, p can examine if the message with the smallest timestamp in *RDVD* passes delivery conditions. If so, the message is delivered and the next one is examined. As presented, DRIFT uses unbounded integers to sequence numbers and timestamps. However, they can be easily bounded by reusing them after some time in a round-robin fashion.

Termination. Observe that for message delivery DRIFT assumes that at least one source continues to multicast messages indefinitely. This assumption can be lifted as follows. If a destination has undelivered messages and has not received a message for a certain time, it floods a dummy message. The delivery of this dummy message is not necessary. The other nodes can use this physical flood to transmit the virtual flooding information required for delivery. Several dummy floods may be required for termination.

Bounding message size. As described, the amount of virtual flooding data appended to each message is proportional to the number of sources in the network. However, the message size or bandwidth limitations may not allow to accomodate all this information in a single message. Observe, however, that the correctness of the algorithm does not depend on the amount of virtual flooding data put into each individual message. Less virtual flooding data per message results in less bandwidth overhead, while potentially larger delivery latency. Note that eliminating the virtual information altogether reduces DRIFT to classic Lamport's TO-multicast [19]. We explore these trade-offs in our simulation.

FIFO and reliable transmission. DRIFT assumes FIFO delivery of messages from single source. However, this assumption is not difficult to implement as the sequence numbers for each message are available. DRIFT may buffer messages received out-of-order and process them in sequence number order. Notice that while the out-of-order messages themselves have to be buffered, the virtual flooding information they carry can be processed without delay. Reliable multicasting is studied extensively in the literature (e.g., [16, 23]). A scheme that detects missed messages and request a retransmission from neighbors or from the source can be easily incorporated into DRIFT.

5 Simulation

For our simulation we use JiST/SWANS v1.0.4 — a simulation environment for ad hoc networks [1, 5]. A distinguishing feature of JiST/SWANS is that it intercepts the calls to the communication layer and dynamically transforms them into calls to the simulator's communication package. Thus, Java applications written for real deployment can be ported to this simulation environment and then placed under a variety of simulation scenarios.

5.1 Setup

Communication between nodes is by broadcast as defined in the 802.11b standard [17]. The communication is subject to interference and message loss. Messages can be lost because of the hidden terminal effect [3] or node disconnects due to mobility. The message loss is modeled using JiST/SWANS' *RadioNoiseIndep* package, which employs a radio model identical to the one used in the popular ns-2 simulator [2]. We simulate a wireless ad hoc network of 100 nodes deployed in a square field of 400×400 meters. We used the tranmission range of 88 meters which is default in JiST/SWANS. The nodes are stationary except for the cases in which we measure the impact of mobility (see Section 5.5). Similarly to the transmission range, we also use the default values of other parameters (such as bandwidth) in JiST/SWANS.

The positions of the nodes in the field are uniformly randomly selected. Sources start TO-multicasting at random times uniformly distributed between 0 seconds and the rate of TO-multicasting. Measurement data is collected only after all nodes have started invoking TO-multicasts. Every source TO-multicasts at least 15 messages with an interval of 100 seconds between messages. The payload size is 128 bytes. Unless otherwise specified, each node is both a source and a destination.

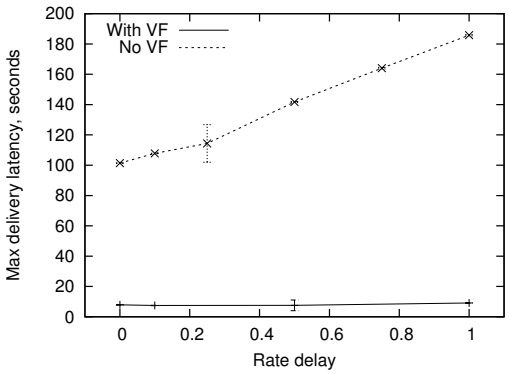
DRIFT uses reliable flooding. To simulate reliable flooding in JiST/SWANS, we set the link packet loss to zero. However, messages are still lost due to hidden terminal effect and node disconnects. To minimize these losses we choose a relatively low flooding rate. Notice that even with this flooding rate and 100 sources, on average, one message per second is TO-multicast. All results are averaged over at least 20 runs in different uniform node distributions. Where significant we indicate the 95%-confidence intervals.

To evaluate the performance of DRIFT, we measure the impact of virtual flooding by comparing the performance of total order multicast flooding with virtual flooding (*Total Order with Virtual Flooding (TOVF)*) and without virtual flooding (*Total Order with Flooding only (TOF)*). In what follows, we measure the delivery latency of TOF and TOVF. That is, the time needed to TO-deliver a message after it is TO-multicast by its source. In our calculations, *speedup* is the latency of TOF divided by the latency of TOVF: $speedup = latency_{TOF}/latency_{TOVF}$. The compared measurements for TOVF and TOF are taken in the same simulation run: for any received message we store the time needed to TO-deliver with and without virtual flooding — and the results are compared for the same source-destination node pairs. In the following, we measure the performance gain through the use of virtual flooding, the impact of the source location, scale and node mobility. Unless explicitly stated otherwise, we use the above default values in our measurements.

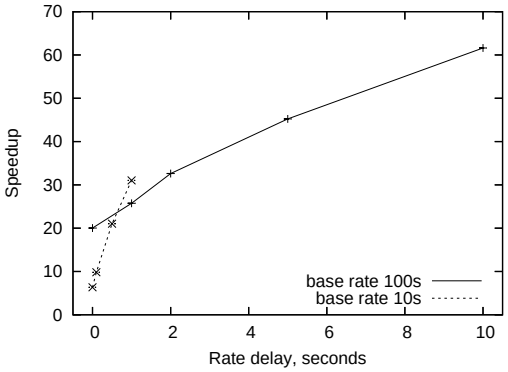
5.2 Speedup: The Impact of Virtual Flooding

The delivery latency of TO-multicast depends on rates with which the sources TO-multicast the messages (called *base rate*) as well as on the relative difference in these rates between the sources. To evaluate the effect of the flooding rate and the relative rate difference, we vary the multicast rate as follows. Node i multicasts with rate $baseRate + i * rateDelay$. We set $baseRate$ at 100 seconds and vary the $rateDelay$. Fig. 4(a) shows the results of these experiments. The y-axis shows the average maximum delivery latency per message. We calculate this latency as follows. For each simulation run we measure the maximum delivery latency for all the messages for a single source. We then compute the average over all sources and all simulation runs. The graph in Fig. 4(a) shows the advantage of TOVF over TOF. This advantage grows as the rate delay increases. Notice that the time to deliver the message by reliable flooding is not shown in Fig. 4(a). However, this delivery latency is in milliseconds and thus negligible compared to the time needed to TO-deliver a message, even using virtual flooding.

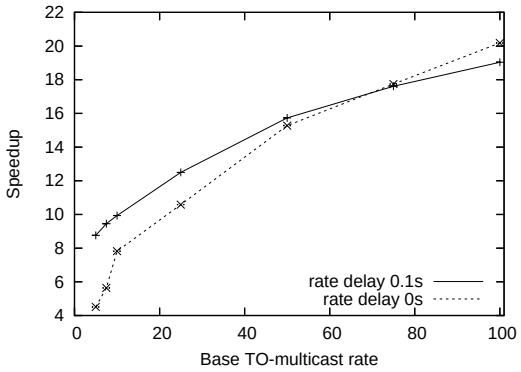
In what follows, the figures only show speedup rather than the absolute performance with and without virtual flooding. Fig. 4(b) shows the speedup for two base rates: 10 and 100 seconds. The speedup increases with increasing rate delay. Also, the speedup is higher with a higher base rate, up to a certain threshold (at a rate delay of approximately 700 milliseconds). Notice that the graph only shows a rate delay interval between 0 and 1 seconds for the base rate of 10 seconds. At 1 second, the lowest rate is an eleventh of the highest rate, similar to the case of base rate of 100 and rate delay of 10 seconds. In Fig. 4(c),



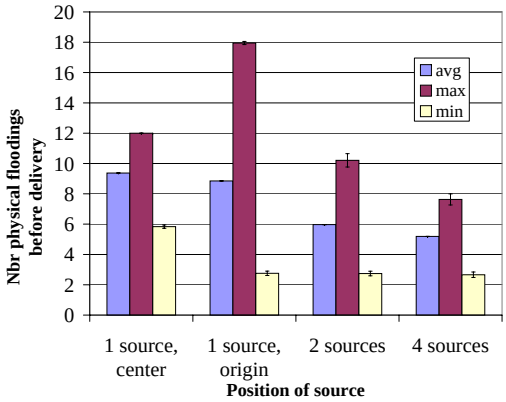
(a) with and without VF: latency as a function of rate delay, seconds



(b) speedup as a function of rate delay



(c) speedup as a function of flooding base rate,



(d) number of floods needed for delivery.

Figure 4. Speedup: delivery latency and number of floods comparison.

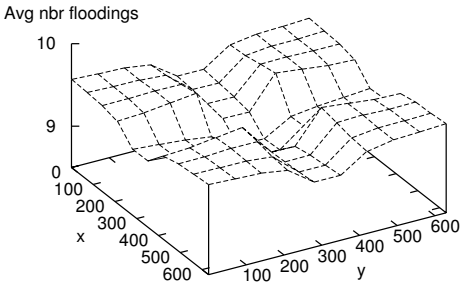
we show the dependency of speedup on the base rate. The rate delay is fixed at 0.1 and 0 seconds. In the latter case all nodes multicast at the same rate. The results show that the speedup is smaller for the curve with 0 compared to the one with 0.1 seconds rate delay, until a base rate of 75 seconds. As the relative impact of the rate delay decreases with increasing base rate, the two curves converge. After a base rate of 75 seconds, the speedup for the curve with 0.1 seconds rate delay is smaller. Because of the rate delay, there may be periods of time during which TO-multicasts are issued within a short time of each other, thus improving the delivery latency for TOF. Hence, the speedup for the curve with 0.1 seconds rate delay is smaller.

5.3 Dependence of Virtual Flooding on Source Locations

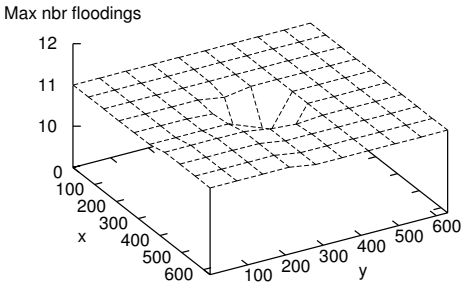
In this subsection we investigate how the positioning of the physical flooding originator affects the efficiency of virtual flooding. We set a fixed number of sources that originate the physical floods (we call them *originating sources*). These nodes invoke TO-multicast. We vary their number and positions in the network. The other sources do not originate the floods, they only use virtual flooding to propagate their messages. Notice that without virtual flooding, no destination is able to TO-deliver messages. There are the total of 1000 messages to send at intervals of 1 second. In this experiment the positions of the nodes are deterministic. 100 nodes are positioned in a 10×10 grid such that each node can only communicate with its adjacent neighbors in the grid (i.e., either having the same x or y coordinate). Direct communication with other nodes is not possible. Notice that the diameter of the network is 18 hops.

We run the simulation for: (i) one originating source is located in the center in the field (position $[315, 315]$, where the first and second number indicate the x-coordinate y-coordinates respectively); (ii) one source is located in the origin (position $[1, 1]$); (iii) two originating sources are located at coordinates $[1, 1]$ and $[629, 629]$; and (iv) four originating sources in every corner of the network. With multiple originating sources, every originating source TO-multicasts one after the other with a fraction of the base rate such that the overall base rate is still 1 second.

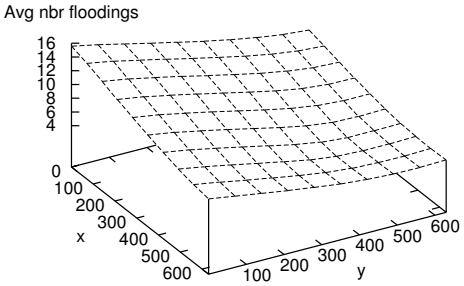
In Fig. 4(d), we show the number of physical floods by originating sources that is needed before all virtually flooded messages



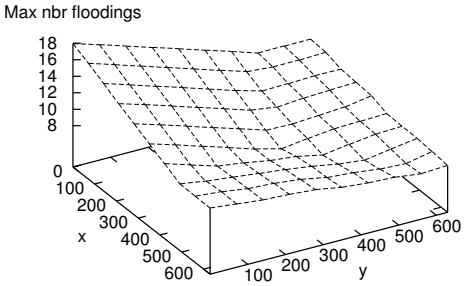
(a) single source positioned at [350,350],



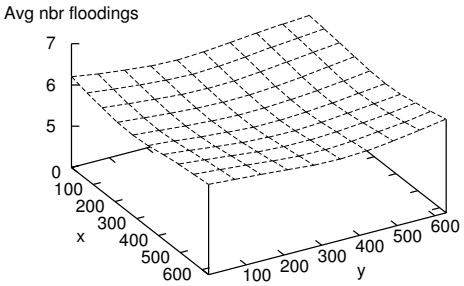
(b) single source positioned at [350,350],



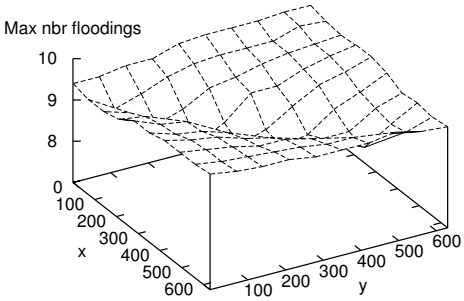
(c) single source positioned at [1,1],



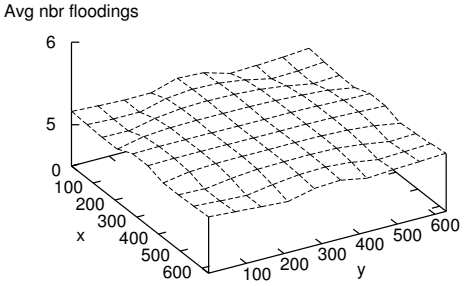
(d) single source positioned at [1,1],



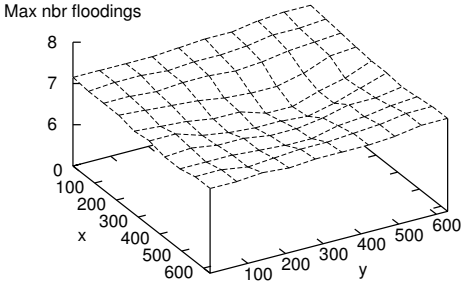
(e) two sources positioned at [1,1] and [629,629],



(f) two sources positioned at [1,1] and [629,629],



(g) four sources in the four corners,



(h) four sources in the four corners.

Figure 5. Source location experiment: 100 nodes positioned on a grid with a single, two, or four originating sources.

are delivered. We show the average, maximum and minimum number of physical floods. The maximum and minimum number are averaged over all simulation runs. The results indicate that, comparing only the single source simulations, the scenario with the originating source located in the center of the network leads to the smaller maximum number of required floods while the source at the origin leads to the smaller average number of required floods. In the first scenario, the maximum is lower because the number of hops between the source and the farthest node is half that for the source at the origin. In the second scenario, the reason for the higher average is more subtle. Indeed, if the source is positioned at the origin, then the delivery latency for a large number of destinations is low because the virtual floodings of other nodes travel with the actual physical flooding (see also the best case scenario in Fig. 1(b)). This is also the reason why the average minimum latency is smaller for this case. Adding more sources decreases the average and maximum delivery latency, but not the minimum delivery latency compared to the source at the origin case.

In Fig. 5, we show the spacial distribution of required floods in our experiment. The x and y-axis show the coordinates of a node, while the z-axis shows the average (in the left column figures) and maximum (in the right column) number of floods needed to deliver a particular message. If the source is positioned at the center, the nodes along the grid line with respect to the physical source have a slightly lower average. Since they are closer (in hop count) to the source than their equivalents on diagonals, they can also deliver messages slightly faster. However, this does not significantly affect the maximum latency (see Fig. 5(b)). If the source is positioned in the origin, the number of required floods is highest close to the source. Indeed, virtual flooding data is traveling slowest in the direction opposite to the physical floods. In this setup, the virtually flooded data has to travel across the network in this least advantageous direction. We also run the simulation for two sources positioned at the origin and at coordinates [629,629] of the field (Figs. 5(e) and (f)), and for four nodes positioned in the corners (sees Fig. 5(g) and (h)). The results are rather intuitive. These results show that the efficiency of virtual flooding and thus DRIFT depends on the position of the sources with low TO-multicast rates relative to the position of sources with high flooding rates.

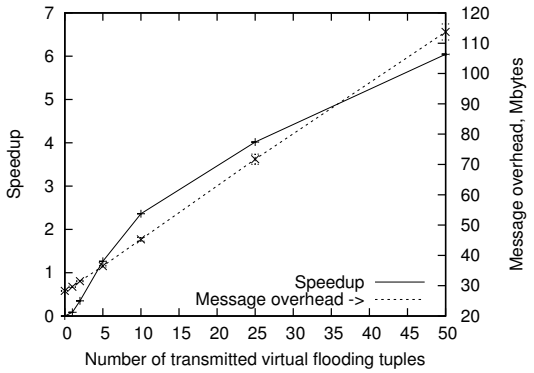


Figure 6. Varying number of virtual flooding entries per physical message.

5.4 Scalability: The Impact of the Number of Sources

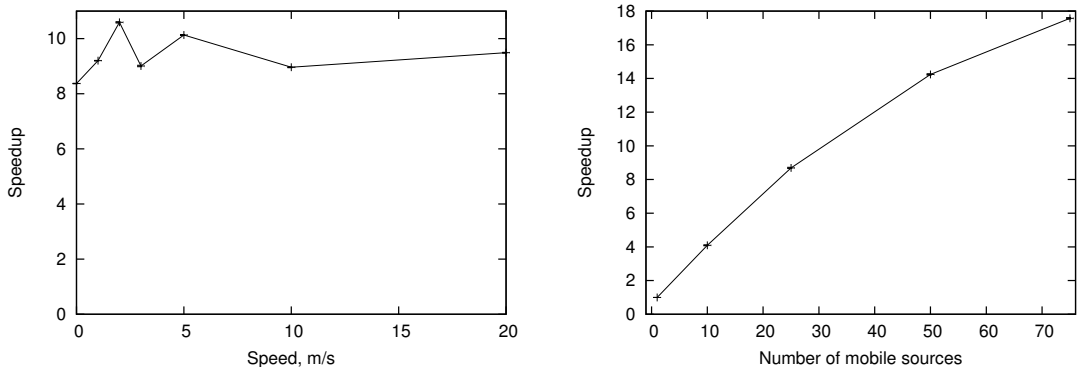
In this experiment we investigate how well DRIFT scales with respect to the number of sources. With the increase of the number of sources, the amount of virtual flooding information each message carries may also increase. This adversely affects the performance of DRIFT. To mitigate this effect, the number of virtual flooding entries per message can be limited. Such limit, however, may delay delivery. We investigate the performance of DRIFT under different limits on the number of flooding entries. The results are shown in Fig. 6. In this graph, the x-axis denotes the limit of virtual flooding entries per message. The maximum limit is the whole network of 50 nodes. All nodes act as sources and TO-multicast every 10 seconds. The y-axis shows the speedup while the second y-axis (indicated by an arrow in the legend of Fig. 6) the message overhead. In the calculation of the overhead, the message payload of 128 bytes is included. The message overhead increases linearly with an increasing amount of virtual flooding information. Interestingly, the speedup increases only sub-linearly with the increased virtual flooding information. Thus, the implementers can select the most appropriate settings for the desired speedup and message overhead.

5.5 The Impact of Mobility

To measure the impact of mobility on delivery latency we use the random way-point model [18]. In our model the speed is fixed and the pause time is zero. This removes the instability caused by varying speeds and pause times.[2] In this model,

[2]Note that it is shown [26] that the random way-point model possesses certain shortcomings. However, for our purposes these shortcomings are unimportant.

each node selects an arbitrary location in the field and moves there in a direct line with constant speed. When it reaches the selected location, it then selects a new location. The flooding base rate is 10 seconds. All other parameters are the same as in the experiment in Fig. 4(a). The results are shown in Fig. 7(a). The results indicate that mobility within the selected speed range does not have significant impact on speedup.



(a) Varying speed, all nodes are mobile. (b)Varying number of sources.

Figure 7. Speedup with mobile sources.

The last experiment studies the effect of the number of sources in the network on speedup. All nodes except for sources are stationary. We vary the number of mobile sources while the overall number of nodes (including sources) stays constant. The results are shown in Fig. 7(b). Interestingly, the speedup increases with increasing number of sources. Thus, virtual flooding, appears to perform better if the number of sources increases.

6 Wireless Sensor Network Implementation

To verify the applicability of DRIFT to practical ad hoc networks, we implemented it on Crossbow's MICA2 motes [15, 14]. The motes are a sensor network platform popular in both academia and industry.

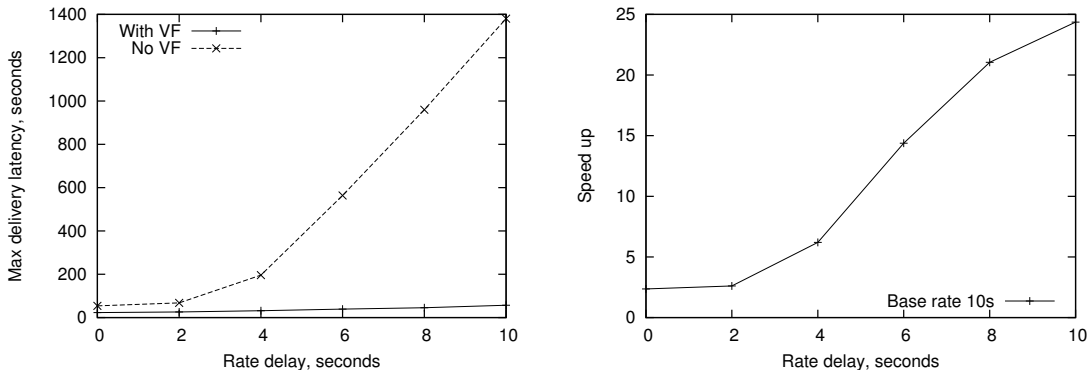
Experimental setup. We used 16 motes arranged in a 4×4 grid. The motes were instrumented with a wired backchannel. The motes run TinyOS v.1.1.15 [14] operating system. As DRIFT assumes single source reliable FIFO delivery, we did not focus on the implementation of this mechanism. To emulate reliable delivery, instead of the radio, the messages are transmitted over the backchannel. Each mote reliably communicates with the adjacent neighbors in the grid. That is, each mote can have up to 4 neighbors and the network's diameter is 6 hops.

To conserve memory and minimize computation overhead on the motes, we implemented the data structure optimizations discussed in Section 4. As the number of messages multicast by each source during the run was known a priori, we further optimized the code. Specifically, we used a two-dimensional array that stored the highest seen timestamp for each source and message sequence number. We implemented TOF and TOVF separately.

The impact of rate delay on latency and speedup. There were 4 sources located in the interior of the grid. Each source multicast 10 messages. Message size was 36 bytes. We used a base rate of 30 seconds. In our experiment we varied the rate delay from 0 to 10 seconds, one measurement was taken at each data point. The results of the experiments are shown in Fig. 8. The observed results coincide with those obtained in the simulation (see Fig. 4). Notice that in our experiments TOVF was up to 20 times faster than TOF. Our experimental results lend greater credibility to the simulation measurements and show the applicability of DRIFT in practice.

7 Handling Dynamic Groups and Failures

So far we assumed that the set of sources is static. However, in some applications, the nodes may join and leave the network. In this case the nodes have to adjust their logical clock entries and other accounting information. Notice that arrival and departure of non-sources does not affect the algorithm. They may simply leave or start TO-delivering messages respectively. In case of sources, the situation is more complicated. If a source intends to leave the network it TO-multicasts a message announcing its departure. It can then immediately leave the network. Upon delivery of this message, the destinations remove this source and adjust their data structures accordingly.



(a) latency as function of rate delay (b) speedup as a function of rate delay

Figure 8. Speedup in implementation: delivery latency comparison.

The procedure of joining the network is as follows. A new source d contacts one of the existing sources (e.g., by using simple, geographically bounded flooding). The existing source then TO-multicasts a join message on d's behalf. Every existing source adds d and updates its data structures accordingly. A special case arises if the network has no existing sources. This special bootstrap case can be handled as described by Cavin et al [9].

Let us now consider crash-faults and un-announced node departures. The latter occurs if the node fails to notify the others when leaving the network. It is handled similar to crashes. Notice again that the crash for a non-source does not affect DRIFT. If a source crashes, the other nodes have to be able to detect this crash. Crash detection can be implemented using simple flooding or other techniques. However, the discussion of fault-detection mechanisms is outside the scope of this paper; the interested reader is referred, for instance, to work of Friedman and Tcharny [13]. Upon detection of a source crash, the detecting source TO-multicasts a message informing the network of the departure of the faulty source.

8 Conclusion

In conclusion, we would like to observe a salient property of DRIFT. While convenient for total order multicast, virtual flooding is applicable to efficient information propagation of any type. For example, time synchronization is frequently required in sensor networks. This application needs to periodically exchange messages between the sensor nodes. The period is rather well defined. Moreover, the time synchronization information is rather compact. Thus, virtual flooding can be used to piggyback other data (e.g. sensor data) on time synchronization messages. Similarly, the virtual flooding required for delivery in DRIFT does not have to be carried by TO-multicast messages only. Any other messages present in the network (e.g. time synchronization messages) can also be suitable. Hence, DRIFT can leverage existing traffic in the network to minimize delivery latency for TO-multicast.

DRIFT and virtual flooding are based on physical flooding as basic communication primitive. However, in other settings message dissemination can be implemented using techniques other than flooding. For example, a minimal connected dominating set [7] or tree-structured routing scheme can be used. DRIFT and virtual flooding can be adapted to work over these topologies as well.

In this paper we studied a TO-multicasting algorithm that is specifically designed to perform well in ad hoc networks. The resultant algorithm acquired new noteworthy qualities. We believe that this is a propitious path of discovery, which may help to re-evaluate other classic solutions for efficient applications in such networks.

Acknowledgments

The authors thank Ken Birman for letting us run our simulations on a computing cluster at Cornell University and M. Kazim Khan of Kent State University for a consultation on statistical aspects of our experiments.

References

[1] JiST/SWANS. http://jist.ece.cs.cornell.edu.
[2] The network simulator - ns-2. http://www.isi.edu/nsnam/ns.
[3] D. Allen. Hidden terminal problems in wireless LAN's. In *IEEE 802.11 Working Group Papers*, 1993.

[4] G. Anastasi, A. Bartoli, and F. Spadoni. Group multicast in distributed mobile systems with unreliable wireless network. In *Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems (SRDS '99)*, pages 14–23, Washington - Brussels - Tokyo, Oct. 1999. IEEE.

[5] R. Barr. *An efficient, unifying approach to simulation using virtual machines*. PhD thesis, Cornell University, Ithaca, NY, 14853, May 2004.

[6] A. Bartoli. Group-based multicast and dynamic membership in wireless networks with incomplete spatial coverage. *Mobile Networks and Applications*, 3(2):175–188, 1998.

[7] V. Bharghavan and B. Das. Routing in ad hoc networks using minimum connected dominating sets. In *Proc. of the Int. Conference on Communications*, Montreal, Canada, June 1997.

[8] K. Birman, A. Schiper, and P. Stephenson. Lightweight causal and atomic group multicast. *ACM Transactions on Computer Systems*, 9(3):272–314, Aug. 1991.

[9] D. Cavin, Y. Sasson, and A. Schiper. Consensus with unknown participants or fundamental self-organization. In *Proc. of the 3rd Int. Conference on AD-HOC Networks & Wireless (ADHOC-NOW)*, pages 135–148, Vancouver, BC, Canada, 2004.

[10] T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *J ACM*, 43(2):225–267, Mar. 1996.

[11] F. Cristian. Asynchronous atomic broadcast. *IBM Technical Disclosure Bulletin*, 33(9):115–116, Feb. 1991.

[12] X. Défago, P. Urbán, and A. Schiper. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Computing Surveys,*, 36(4):372–421, Dec. 2004.

[13] R. Friedman and G. Tcharny. Evaluating failure detection in mobile ad-hoc networks. *Int. Journal of Wireless and Mobile Computing*, 1(8), 2005.

[14] J. Hill and D. Culler. Mica: A wireless platform for deeply embedded networks. *IEEE Micro*, 22(6):12–24, Nov./Dec. 2002.

[15] J. Hill, R. Szewczyk, A. Woo, D. Culler, S. Hollar, and K. Pister. System architecture directions for networked sensors. *ACM SIGPLAN Notices*, 35(11):93–104, Nov. 2000.

[16] J. W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 81–94, New York, NY, USA, 2004. ACM Press.

[17] IEEE. 802.11 specification (part 11): Wireless LAN medium access control (MAC) and physical layer (PHY) specifications, June 1997.

[18] D. Johnson and D. Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[19] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[20] J. Luo, P. T. Eugster, and J.-P. Hubaux. PILOT: ProbabilistIc Lightweight grOup communication sysTem for Mobile Ad Hoc Networks. *IEEE Transactions on Mobile Computing*, 3(2):164–179, 2004.

[21] N. Malpani, Y. Chen, N. H. Vaidya, and J. L. Welch. Distributed token circulation in mobile ad hoc networks. *IEEE Transactions on Mobile Computing*, 4(2):154–165, 2005.

[22] S. Navaratnam, S. Chanson, and G. Neufeld. Reliable group communication in distributed systems. In *Proc. of the 8th Int. Conference on Distributed Computing Systems (ICDCS'88)*, pages 439–446, San Jose, CA, USA, 1988.

[23] K. Obraczka, K. Viswanath, and G. Tsudik. Flooding for reliable multicast in multi-hop ad hoc networks. *Wireless Networks: The Journal of Mobile Communication, Computation and Information*, 7(6):627–634, 2001.

[24] R. Prakash, M. Raynal, and M. Singhal. An efficient causal ordering algorithm for mobile computing environments. In *ICDCS '96: Proceedings of the 16th International Conference on Distributed Computing Systems (ICDCS '96)*, pages 744–751, Washington, DC, USA, 1996. IEEE Computer Society.

[25] A. Schiper, J. Eggli, and A. Sandoz. A new algorithm to implement causal ordering. In J.-C. Bermond and M. Raynal, editors, *3rd International Workshop on Distributed Algorithms*, volume 392 of *Lecture Notes in Computer Science*, pages 219–232, Nice, France, 26–28 Sept. 1989. Springer.

[26] J. Yoon, M. Liu, and B. Noble. Random waypoint considered harmful. In *INFOCOM 2003*, Apr. 2003.