Experimental Evaluation of Drift: Total Message Ordering in Ad Hoc Networks

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

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

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

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

Technical Report TR-KSU-CS-2006-5
July 4, 2006

Abstract

We present DRIFT[1], a total ordering multicast algorithm optimized for ad hoc networks. DRIFT combines virtual flooding with a communication history ordering algorithm. Virtual flooding is a way of using unrelated message streams to propagate message causality information in order to accelerate message delivery. We review the total order multicast problem and the unique challenges posed in ad hoc networks. We describe DRIFT and its use of virtual flooding in detail. We provide optimizations for implementation. We implement DRIFT on a wireless sensor network. For comparison, we implement the communication history ordering algorithm, TOF, on which DRIFT is based. Both require that all messages be received by all nodes, we implement a reliable local broadcast scheme to achieve reliable message transmission. We evaluate DRIFT's performance through experimentation and simulation. We employ wireless channel emulation as well as radio transmission in our experimental approach. We study the effects of varying the relative rate at which messages are multicast using both physically flooded messages and non-flooded messages as carriers of the virtual flood. We examine the impact of varying the period of non-flooded messages. From our wireless transmission experiments we find DRIFT delivers multicast messages twice as fast as TOF using only physically flooded messages as carriers. DRIFT achieves a sixfold speedup when periodic non-flooded messages act as carriers in addition to the physically flooded messages.

Keywords: Ad hoc networks, total order multicast, wireless sensor networks, virtual flooding.

Corresponding author: Thomas Clouser

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

[1]DRIFT was originally described by Stefan Pleisch, Thomas Clouser, Mikhail Nesterenko and André Schiper. [1]

1 Introduction

Recent advances in PDA and wireless sensor technology enable 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 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 [2]. 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.

In implementation, to guarantee that all nodes receive all messages, flooding must be augmented with a *reliable local broadcast* scheme. Such schemes may make use of periodic non-flooded messages to recover lost messages. These messages provide another carrier for a virtual flood. By utilizing these non-flooded messages, DRIFT can propagate message causality information faster, leading to lower delivery latency. We present simulation and experimentation 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 describe implementation considerations for DRIFT. In Section 6 we describe our implementation of DRIFT in a wireless sensor network and present results obtained from experimentation and simulation. We conclude the paper in Section 7.

2 Total Order Multicast and Ad Hoc Networks

Total order multicast (or TO-multicast)[2] 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. In their survey paper, Défago et al. [3] classify the algorithms according their ordering techniques: *sequencer-based, privilege-based, destination*, and *communication history*.

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

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 survey paper [3]. 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. [4]. Anastati et al. [5] 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.

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 [7]. A token-based algorithm in mobile ad hoc networks is described by Malpani et al. [8]. 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 [9]. 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 [2] 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 [10, 2, 11]. Prakash et al. [12] 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. It also scales well as there is no need for

extra ordering messages. DRIFT belongs to this class.

Probabilistic multicast. Luo et al. [13] 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 [2, 3] (which is the basis of DRIFT). This algorithm assumes FIFO communication channels and reliable message transmission. 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. 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 Figure 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 Figure 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 Figure 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.

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 chapter.

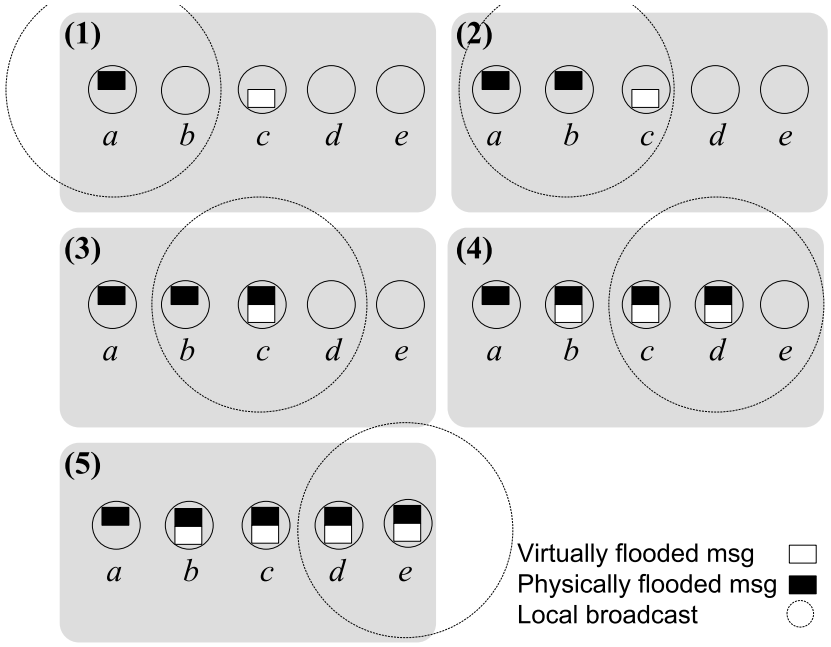
4 DRIFT Description

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 chapter by describing an extension for using non-flooded messages as carriers for virtual flooding.

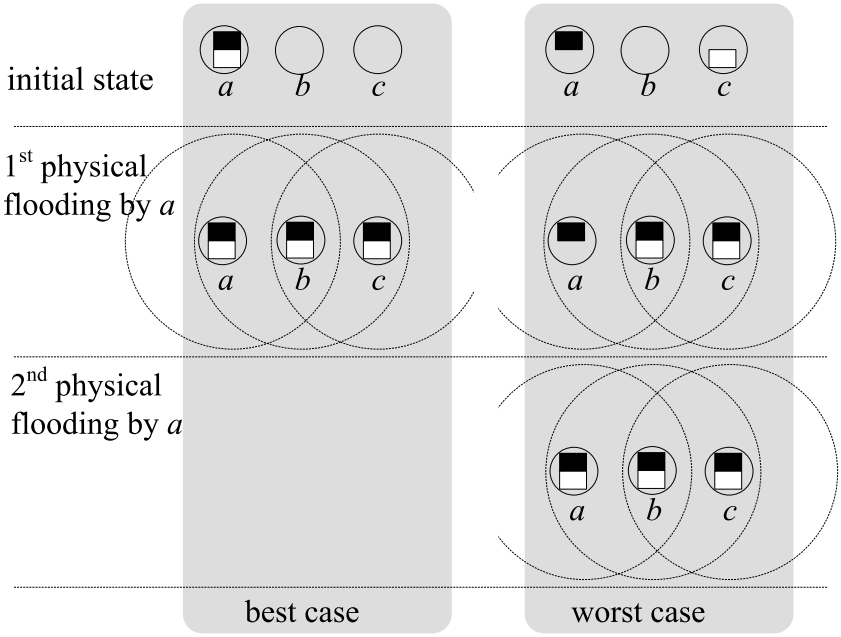
Initially, we assume that destinations are static. Each flooded message is reliably received by every node. Multiple messages from an individual source are received by each node in FIFO order. Nodes do not crash. The sources do not join or leave the network (i.e. we consider *static* group membership). Furthermore, we assume that at least one source sends an infinite number of messages. We discuss how these assumptions may be relaxed or implemented in Chapter 5.

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



(i) Virtual flooding example.



(ii) Efficiency of virtual flooding.

Figure 1: Virtual flooding example.

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 Chapter 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 sent 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 Figure 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 Figure 2): virtual flooding update (*vf update*), received message processing (*rcpt processing*), and message delivery

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.

(*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: $RcvdSN[i] = isn$ 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 Figure 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_3, while b multicasts m_2. In our example we focus on the delivery of the messages at destinations c and d and skip unrelated events. The depicted events happen in sequence. The sequence is from top to bottom.

4.4 Extension for Non-Flooded Messages

In addition to the physically flooded multicast messages, we may employ non-flooded messages as carriers of the virtual flood. By using these messages as carriers we can improve the latency by increasing the speed at which message causality information is propagated throughout the network. Notice that sending and receiving these messages should not modify the value of the node's logical clock or sequence number.

The additional actions required for each node p to use non-flooded messages is shown in Figure 4. **sendNonFlooded** is invoked when the node is ready to broadcast a non-flooded message. When p receives a non-flooded message, the second action — **receiveNonFlooded**, is executed.

When a node p has a non-flooded message to broadcast, it executes **sendNonFlooded**. This action attaches the freshest virtual flooding information to the non-flooded message. When a node receives a non-flooded message messages, the message *qm* is processed, the *Seen* is updated and, if the node is a destination, *RCVD* is checked for deliverable messages.

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.

5 Implementation Considerations

Many of the assumptions made in our description of DRIFT are non-trivial to implementing. For example, it is not uncommon for nodes to be mobile, limited in resources or prone to failure. We now discuss various ways of relaxing the assumptions.

5.1 Optimizing Data Structures

Some of the wireless ad hoc platforms have limited memory resources (e.g. Crossbow's MICA2 motes [14]). The data structures used in DRIFT can be optimized to reduce memory consumption at each individual node. 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.

node p

actions

sendNonFlooded(m)

broadcast$(m,$ **getHighestTimestamp**$(\mathit{Seen}))$

when receiveNonFlooded$(\mathit{qm}, \mathit{qSeen})$

processMessage(qm)

vf update: $\mathit{Seen} := \mathit{Seen} \cup \mathit{qSeen}$

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

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

$\mathit{DLVD} := \mathit{DLVD} \cup \mathit{READY}$

while $\mathit{READY} \neq \varnothing$ **do**

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

TO-deliver *vm*

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

Figure 4: DRIFT additional actions.

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 received message $\mathit{RcvdSN}[q]$. This is the entry that is used in case the node gets virtual flooding data that there is an outstanding message from 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 *RCVD* 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.

5.2 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 multicasts 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 multicasts may be required for termination.

5.3 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 accommodate 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 [2].

5.4 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.

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 [15].

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 [16]. Upon detection of a source crash, the detecting source TO-multicasts a message informing the network of the departure of the faulty source.

5.5 FIFO and Reliable Transmission

DRIFT assumes FIFO delivery of messages from a 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 message receipt is required for DRIFT. Message reception rates in wireless networks are significantly lower than they are in wired networks. Furthermore flooding oversaturates the channel, causing deterioration of message reception [17]. Even though a node is given the opportunity to receive a single message multiple times, due to collisions the node may not receive the message altogether [18].

An effective mechanism of ensuring reliable message receipt is *reliable local broadcast*. In reliable broadcast every neighbor of the sender receives the message. Notice that even though wireless radio transmissions are broadcasts, reliability is not guaranteed as some of the nodes may not receive the message. Several methods of implementing reliable broadcast are described in the literature [19, 20, 21, 22, 23].

In Alagar et al.[19] each node maintains a history of messages sent and received. When a node detects a new neighbor, they exchange history information. Each node then transmits messages that the other is missing. Acknowledgments are used to confirm message receipt. Pagani [20] present a protocol that assumes the hosts are grouped into clusters. Each cluster head is responsible for ensuring reliable local delivery within its cluster. The cluster head of the cluster that originated the message, is responsible for ensuring all clusters successfully receive the message. In RBS [21] receivers detect the loss of a message through *relayer broadcast sequence numbers* from their neighbors. When a loss is detected, a negative acknowledgment, NACK, is sent to the neighbor. Upon receipt of the notification, the neighbor rebroadcasts the lost message. In addition to NACKs, ROB [22] uses the detection of collisions by one-hop neighbors to cause the retransmission of a message.

Livadas and Lynch [23] describe a reliable broadcast scheme based on frontier messages. Their scheme is particularly suitable for resource contrained devices. Each node does not have to maintain a neighborhood set, large buffers for message reordering or any other data. The frontier message algorithm works as follows. The source attaches a sequence number to each message it sends. The receiver keeps track of the messages received in FIFO order. Let hsn be the highest sequence number among such messages. Periodically, each receiver broadcasts a *frontier message* containing its hsn value. If a node receives this message with an hsn smaller than its local value, the receiver rebroadcasts the missing messages. Thus, eventually all nodes receive all source messages.

6 Wireless Sensor Network Implementation

In this chapter we discuss the implementation of DRIFT for a wireless sensor network, the challenges we encounter, and our experimental and simulation approach. In Section 6.1 we describe our target ad hoc network platform. We solve many challenges to implement DRIFT for a wireless sensor network, we discuss our solutions to these challenges in Section 6.2. In Section 6.3, we define our approach for evaluating DRIFT's performance. We present our experimental and simulation results in Section 6.4 and Section 6.5, respectively.

6.1 Implementation Platform

To evaluate the performance of DRIFT in a practical ad hoc network we implement it to run on Crossbow's MICA2 motes [14, 24]. The motes are a sensor network platform popular in both academia and industry. They run the TinyOS [24] operating system. We use the nesC [25] programming language to implement DRIFT.

We perform our experiments on a static testbed we call *BenchNet*. BenchNet contains 32 MICA2 motes mounted to a fixed surface. Each mote is connected via its communication port to an ethernet programming board. These programming boards form the instrumental backchannel that allow monitoring applications and the motes to communicate.

Our simulations are run in TOSSIM [26], a wireless sensor network simulator that is source code compatible with the MICA2 platform. Thus, we are able to use the same code for our experiments and simulations. This gives greater fidelity to our simulation results.

6.2 Implementation Challenges

DRIFT and the total order multicast algorithm on which it is based require reliable message transmission. Achieving reliable message transmission is the foremost challenge in implementing DRIFT on our chosen platform. One approach is to replace the wireless broadcasts with *wireless channel emulation*. When a mote is to multicast a message, the message is sent over the instrumental backchannel to the wireless channel emulator. The emulator then sends the message to nodes that should receive it based on predefined rules. This approach provides greater control over the message propagation at the expense of lower fidelity to the actual wireless signal propagation. A second approach is to implement in-network reliability with a mechanism such as reliable local broadcasts. This approach provides higher fidelity but results in uncontrollable experiment running times as network density increases. To bound the running time, the scale of the experiments has to be limited. We perform experiments using both approaches.

Single source FIFO message delivery is also required by DRIFT. In environments where memory resources are plentiful, a reordering buffer would be appropriate. When memory is scarce, the simplest solution is to drop out-of-order messages. When destination r receives a message from source q, r

drops the message if *qsn* $\neq$ *RcvdSN*$[q] + 1$. This approach may incur additional messaging overhead. It is the one we take.

Wireless sensor network devices are characterized by their scarcity of memory and computational resources. MICA2 motes are no exception. Similar to many embedded environments, TinyOS only permits static memory allocation. To conserve memory and minimize computation overhead, our implementations use the following data structure optimizations. As the number of source nodes and the number messages that the source nodes will multicast is known a priori, we can limit the size of *Seen*, *RCVD*, *DLVD* and *RcvdSN*. *Seen* becomes a two dimensional array that maintains the entry with the highest timestamp seen for a particular source and sequence number. The *RCVD* set only needs to be large enough to hold a sufficient number of undelivered messages to allow progress. By limiting the number of messages multicast by each source and the maximum rate delay (described in Section 6.3), we can greatly reduce the size of *RCVD*. We avoid the additional memory overhead maintaining a *READY* set by providing a function to retrieve the minimum message in *READY*. We use *RcvdSN* to deterimine if a message arrives out-of-order.

When using frontier messages, each node must maintain a message history. To minimize its size, we only maintain information necessary to reconstruct lost messages. That is, for each source and sequence number we record the logical clock value.

6.3 Performance Evaluation Approach

The delivery latency of TO-multicast depends on the rates at which the sources TO-multicast messages, the *base rate*, as well as the relative difference in these rates between the sources, the *rate delay*. To evaluate the effect of the flooding rate and the relative rate differences, we vary the multicast rate as follows. Source i multicasts with rate $baseRate + i \times rateDelay$. Base rates are chosen to ensure sufficient time for all nodes to receive all messages multicast when there is no rate delay.

We measure the impact of virtual flooding by comparing the performance of total order multicast 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 has been received at a destination node. In our calculations, *speedup* is the latency of TOF divided by the latency of TOVF: $speedup = latency_{TOF} / latency_{TOVF}$. In addition to measuring the delivery latency of TOF and TOVF, we study the impact of using the frontier messages as a carrier for virtual flooding (*Total Order with Virtual Flooding Plus* (TOVF+)) in Subsection 6.4.2. When comparing TOF to TOVF+ speedup is: $speedup = latency_{TOF} / latency_{TOVF+}$. Unless noted otherwise, the measurements for TOF and TOVF or TOVF+ are taken in the same experimental trial — for any received messages we store the time needed to TO-deliver with and without virtual flooding.

6.4 Experimental Results

6.4.1 Wireless Channel Emulation

Setup. We arrange 16 motes in a 4×4 grid. Each mote reliably communicates with the adjacent neighbors in the grid. That is, each mote has up to 4 neighbors and the network's diameter is 6 hops. We implement TOF and TOVF separately. The 4 interior nodes are sources. All nodes are destinations. Each source multicasts 10 messages. We use a base rate of 30 seconds. In our experiment we vary the rate delay from 0 to 10 seconds, and we take one measurement at each data point.

Results. Figure 5 shows the results of these experiments. In Figure 5(a), the y-axis shows the average maximum delivery latency. At each rate delay we measure the maximum delivery latency of all messages from a source. We then average over all sources. The speedup of TOVF compared to TOF is shown in Figure 5(b). The graphs indicate that in our experiment TOVF can deliver messages up to 20 times faster than TOF.

6.4.2 Wireless Transmission

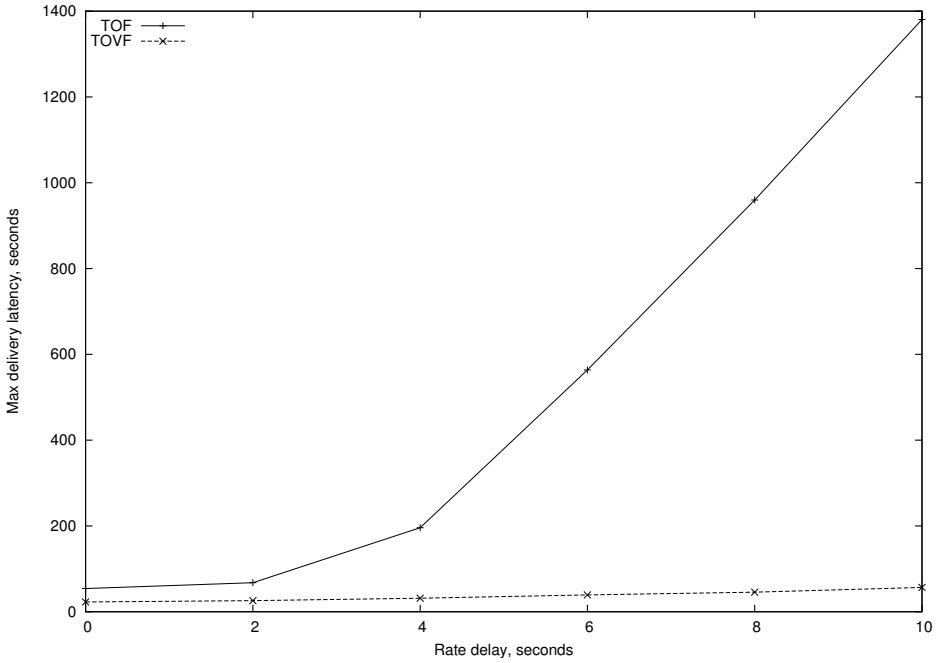
Setup. We arrange 5 motes in a line. We place each mote so that it can reliably communicate with its adjacent neighbors, but cannot communicate with other motes. That is, each mote has up to 2 neighbors and the network's diameter is 4 hops. Each node is a source and a destination. Source nodes multicast 10 messages. We use a base rate of 25 seconds. We vary the rate delay from 0 to 7 seconds. We fix the frontier message rate at 6 seconds. That is, each node will send a frontier message every 6 seconds.

Results. Figure 6 shows the results of these experiments. In Figure 6(a), the y-axis shows the average maximum delivery latency. This latency is calculated as follows: for each experimental trial we measure the maximum delivery latency of all messages from a source. We then compute the average over all sources and at least 5 trials. The speedup of TOVF and TOVF+ compared to TOF is shown in Figure 6(b).

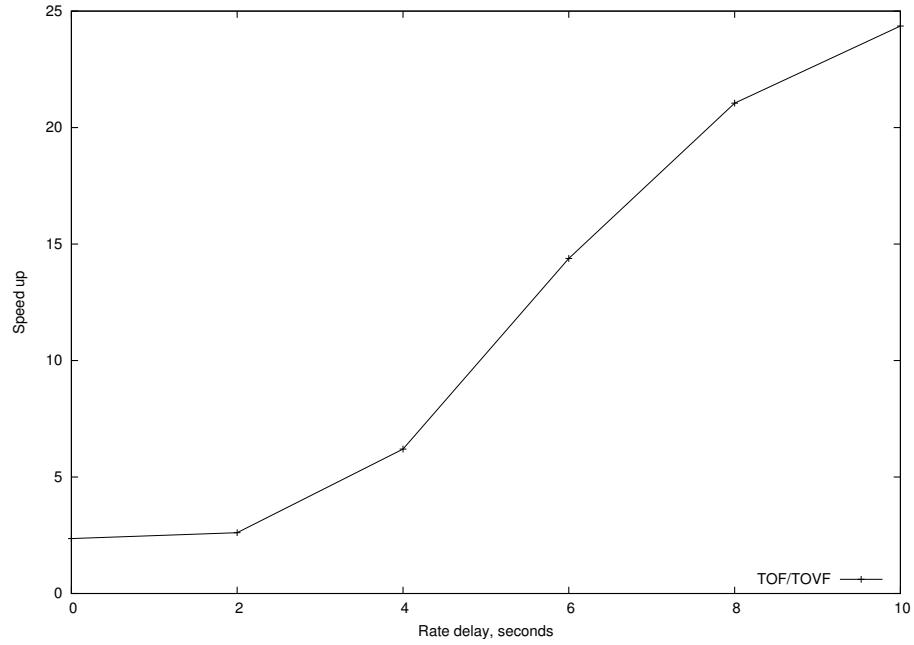
The graphs in Figure 6 show that both TOVF and TOVF+ outperform TOF. Notice the TOVF+ speedup increases with rate delay while the TOVF speedup remains constant. This is due to the following. For TOVF the highest timestamped entries from *Seen* are carried in the original flooding of the source message and any rebroadcasts of the source message. With TOVF+ these entries are sent with each frontier message as well, increasing the speed at which *Seen* gets populated with the entries necessary to TO-deliver the next message in *READY*. By sending and receiving the highest timestamped entries more frequently, TOVF+ is able to achieve a greater speedup than TOVF. As TOVF+ does not respond to changes in rate delay, its scalability should be higher.

6.4.3 Impact of Frontier Message Rate

Setup. We conduct these experiments in the same manner as in Subsection 6.4.2. We fix the rate delay at 0 seconds, use a base rate of 25 seconds and vary the frontier message rate from 0 to 30 seconds.

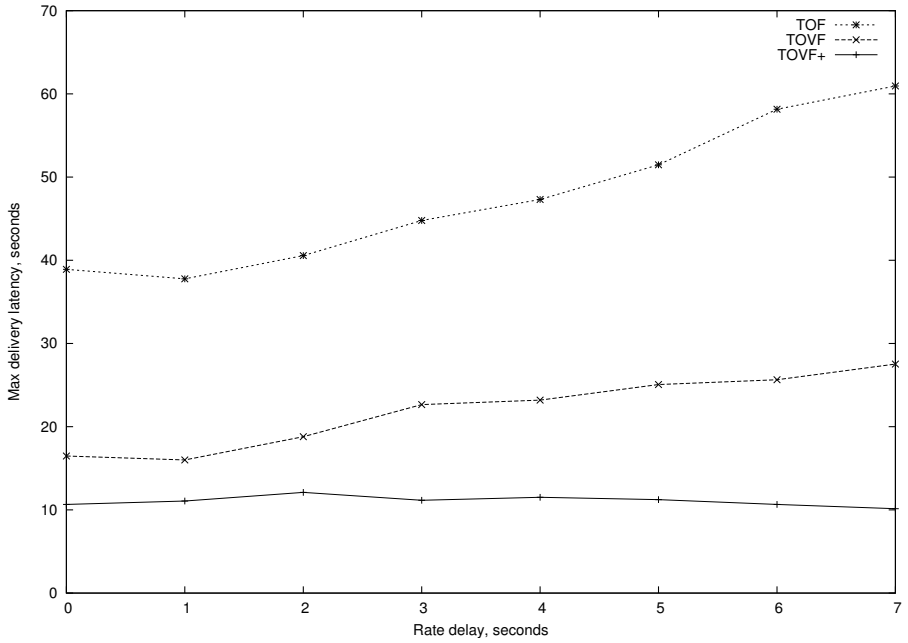


(a) latency as function of rate delay

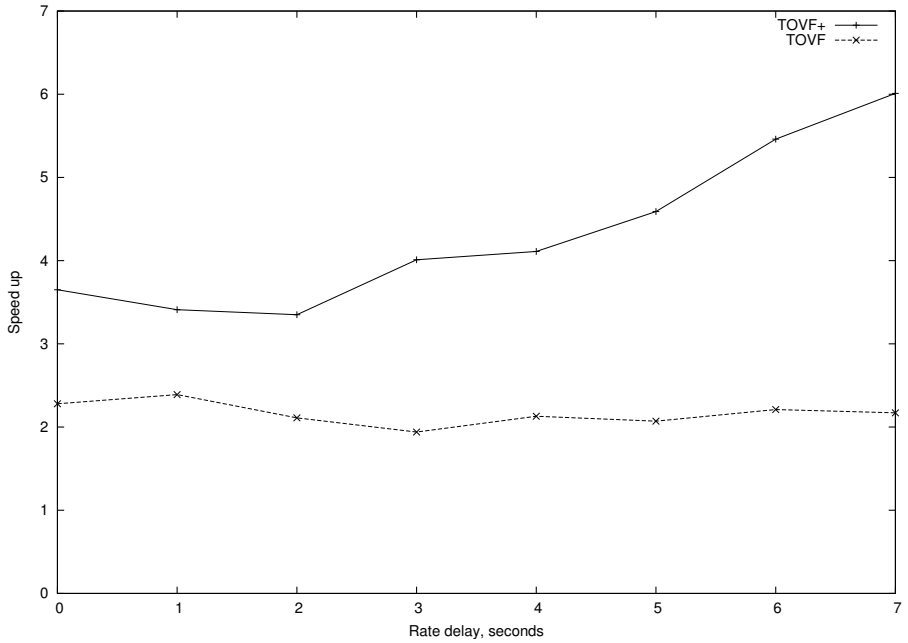


(b) speedup as a function of rate delay

Figure 5: Speedup in implementation: wireless channel emulation.



(a) latency as function of rate delay



(b) speedup as a function of rate delay

Figure 6: Speedup in implementation: wireless transmission.

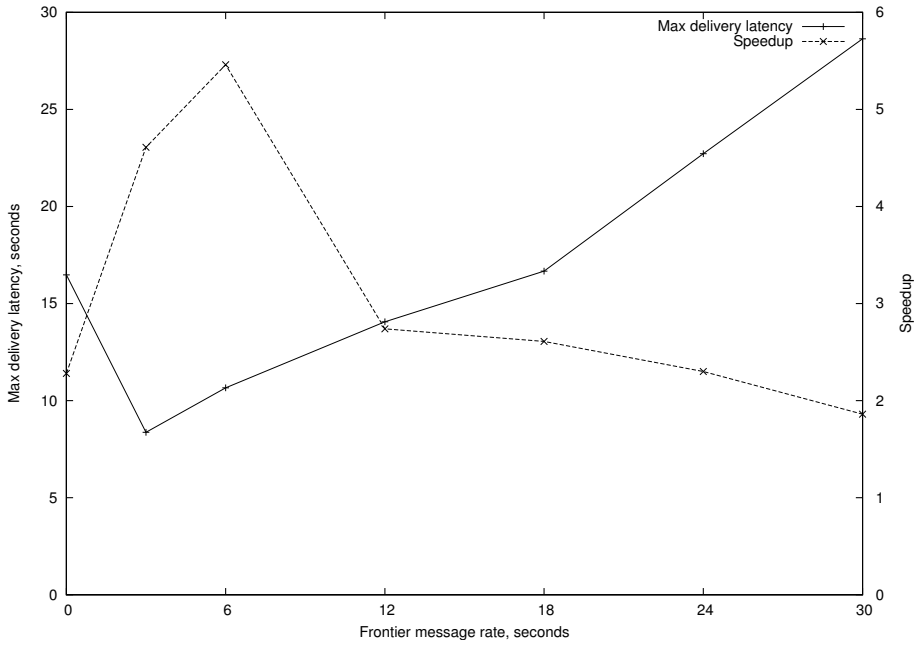


Figure 7: Impact of frontier message rate.

Results. Figure 7 shows the results of these experiments. The y-axis shows the maximum delivery latency of TOVF+ while the second y-axis shows the speedup compared to TOF. Note that a frontier message rate of 0 is TOVF. The graph in Figure 7 shows that the frontier message rate has a significant impact on both the maximum delivery latency and the obtained speedup. It should be noted that the rate at which frontier messages are sent impacts both the recovery speed for lost messages and the speed at which causality information propagates.

As there is a range of frontier message rates at which TOVF+ outperforms TOVF, one can tailor the rate to meet the system objectives. If sufficient channel bandwidth exists to handle the additional traffic of sending frontier messages frequently, then the lowest possible delivery latencies can be achieved. If the additional overhead is unacceptable, the system may still achieve lower delivery latencies than TOVF while sending fewer frontier messages.

6.5 Simulation Results

Setup. We simulate 50 motes arranged in 5×10 grid. Each mote can communicate with its adjacent neighbors in the grid. That is, each mote has up to 4 neighbors and the network's diameter is 13. The links are reliable. The 4 nodes located on the corners of the grid are sources. All nodes are destinations. The source nodes multicast 10 messages. We use a base rate of 100 seconds. In our simulation we vary the rate delay from 0 to 7 seconds. We fix the frontier message rate at 6 seconds.

Results. The graph in Figure 8 shows that both TOVF and TOVF+ outperform TOF in the simulation, with TOVF+ achieving a greater speedup than TOVF. Our simulation result concur with our

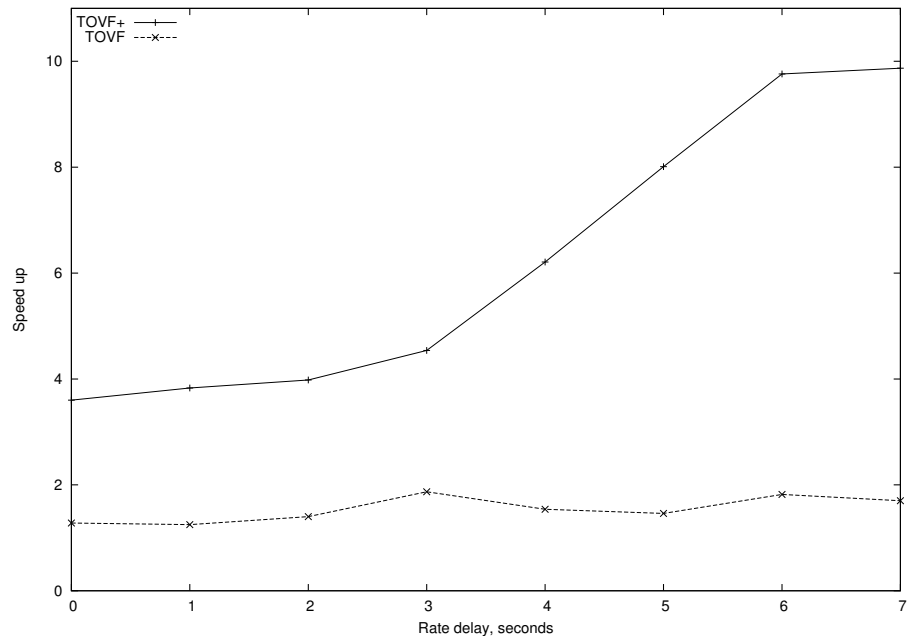


Figure 8: Speedup as a function of rate delay.

experimental results.

7 Conclusion

We would like to observe a salient property of DRIFT. While convenient for total order multicast, virtual flooding can efficiently propagate information of any type. In wireless sensor networks there are many commonly used services that would provide the carriers for virtually flooded data. For example, time synchronization is frequently required in sensor networks. This service needs to exchange messages between the sensor nodes at a fixed rate. Moreover, the time synchronization information is rather compact. Thus, virtual flooding can be used to propagate other data (e.g. sensor data) on time synchronization messages.

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 [27] or tree-structured routing scheme can be used. DRIFT and virtual flooding can be adapted to work over these topologies as well.

We demonstrated through experimentation and simulation the effectiveness of DRIFT as a total order multicast delivery mechanism for ad hoc networks. Future work calls for more detailed exploration of the applicability of DRIFT. In particular, it would be beneficial to understand the performance gains achievable on resource rich platforms, in field deployments, and at much greater scale.

References

[1] S. Pleisch, T. Clouser, M. Nesterenko, and A. Schiper, "Drift: Efficient message ordering in ad hoc networks using virtual flooding," Tech. Rep. LSR-REPORT-2006-002, Swiss Federal Institute of Technology (EPFL) Distributed Systems Laboratory (LSR), May 2006.

[2] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, pp. 558–565, July 1978.

[3] X. Défago, P. Urbán, and A. Schiper, "Total order broadcast and multicast algorithms: Taxonomy and survey," *ACM Computing Surveys*, vol. 36, pp. 372–421, Dec. 2004.

[4] 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)*, (San Jose, CA, USA), pp. 439–446, 1988.

[5] 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)*, (Washington - Brussels - Tokyo), pp. 14–23, IEEE, Oct. 1999.

[6] A. Bartoli, "Group-based multicast and dynamic membership in wireless networks with incomplete spatial coverage," *Mobile Networks and Applications*, vol. 3, no. 2, pp. 175–188, 1998.

[7] F. Cristian, "Asynchronous atomic broadcast," *IBM Technical Disclosure Bulletin*, vol. 33, pp. 115–116, Feb. 1991.

[8] N. Malpani, Y. Chen, N. H. Vaidya, and J. L. Welch, "Distributed token circulation in mobile ad hoc networks," *IEEE Transactions on Mobile Computing*, vol. 4, no. 2, pp. 154–165, 2005.

[9] T. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *J ACM*, vol. 43, pp. 225–267, Mar. 1996.

[10] K. Birman, A. Schiper, and P. Stephenson, "Lightweight causal and atomic group multicast," *ACM Transactions on Computer Systems*, vol. 9, pp. 272–314, Aug. 1991.

[11] A. Schiper, J. Eggli, and A. Sandoz, "A new algorithm to implement causal ordering," in *3rd International Workshop on Distributed Algorithms* (J.-C. Bermond and M. Raynal, eds.), vol. 392 of *Lecture Notes in Computer Science*, (Nice, France), pp. 219–232, Springer, 26–28 Sept. 1989.

[12] 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)*, (Washington, DC, USA), pp. 744–751, IEEE Computer Society, 1996.

[13] 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*, vol. 3, no. 2, pp. 164–179, 2004.

[14] J. Hill, R. Szewczyk, A. Woo, D. Culler, S. Hollar, and K. Pister, "System architecture directions for networked sensors," *ACM SIGPLAN Notices*, vol. 35, pp. 93–104, Nov. 2000.

[15] 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)*, (Vancouver, BC, Canada), pp. 135–148, 2004.

[16] R. Friedman and G. Tcharny, "Evaluating failure detection in mobile ad-hoc networks," *Int. Journal of Wireless and Mobile Computing*, vol. 1, no. 8, 2005.

[17] Y.-C. Tseng, S.-Y. Ni, Y.-S. Chen, and J.-P. Sheu, "The broadcast storm problem in a mobile ad hoc network," *Wirel. Netw.*, vol. 8, no. 2/3, pp. 153–167, 2002.

[18] 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*, vol. 7, no. 6, pp. 627–634, 2001.

[19] S. Alagar, S. Venkatesan, and J. Cleveland, "Reliable broadcast in mobile wireless networks," in *In Military Communications Conference, MILCOM Conference Record*, pp. 236–240, 1995.

[20] E. Pagani, "Providing reliable and fault tolerant broadcast delivery in mobile ad-hoc networks," *Mob. Netw. Appl.*, vol. 4, no. 3, pp. 175–192, 1999.

[21] S. Y. Cho, J. H. Sin, and B. I. Mun, "Reliable broadcast scheme initiated by receiver in ad hoc networks," *lcn*, vol. 00, p. 281, 2003.

[22] S. Park and R. R. Palasdeokar, "Reliable one-hop broadcasting (rob) in mobile ad hoc networks," in *PE-WASUN '05: Proceedings of the 2nd ACM international workshop on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks*, (New York, NY, USA), pp. 234–237, ACM Press, 2005.

[23] C. Livadas and N. A. Lynch, "A reliable broadcast scheme for sensor networks," Tech. Rep. MIT-LCS-TR-915, Massachusetts Institute of Technology Computer Science and Artificial Intelligence Laboratory, August 2003.

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

[25] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesc language: A holistic approach to networked embedded systems," in *PLDI '03: Proceedings of the ACM*

SIGPLAN 2003 conference on Programming language design and implementation, (New York, NY, USA), pp. 1–11, ACM Press, 2003.

[26] P. Levis, N. Lee, M. Welsh, and D. Culler, "Tossim: accurate and scalable simulation of entire tinyos applications," in *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, (New York, NY, USA), pp. 126–137, ACM Press, 2003.

[27] 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.