Group Multicast in Distributed Mobile Systems with Unreliable Wireless Network

Giuseppe Anastasi, Alberto Bartoli*, Francesco Spadoni
Dept. of Information Engineering
University of Pisa, Italy
e-mail: anastasi@iet.unipi.it

**Dipartimento di Elettrotecnica, Elettronica ed Informatica*
University of Trieste
e-mail: bartolia@univ.trieste.it

Abstract

In this paper we propose a multicast protocol for a distributed system that includes mobile hosts. The protocol guarantees reliable delivery, i.e. delivery of every multicast and absence of duplicates. The sender of each multicast may select among three increasingly strong delivery ordering guarantees: FIFO, Causal, Total. We make loose assumptions on the underlying computing system: (i) we consider an unreliable wireless network, i.e. one that provides only incomplete spatial coverage and such that messages could be lost even within cells (e.g., due to physical obstructions); (ii) movements are unpredictable, i.e. a user that leaves a cell may enter any other cell, perhaps after a potentially long disconnection. Our solution does not store any sensible state information at mobile support stations, thus movements do not trigger the transmission of any message in the wired network and no notion of hand-off is used. Furthermore, movements at inopportune times can cause only occasional performance penalty but do not affect correctness. Weak assumptions on the underlying computing system, absence of state information at mobile support stations and loose mobility assumptions, contribute to improve the reliability of applications deployed over the proposed protocol.

1. Introduction

Multicast communication is a powerful paradigm for structuring distributed programs in which multiple processes must closely cooperate together [6,13]. Supporting such paradigm in the context of mobile computing poses a number of problems not present in "stationary" distributed computing, due primarily to resource constraints on mobile devices and limited bandwidth of wireless links [10]. Furthermore, mobility could cause loss of multicasts even in the absence of failures [1]. In this paper we propose a multicast protocol for a distributed system in which mobile hosts communicate with a wired infrastructure through a number of spatially limited cells that define wireless links. Our solution is flexible and provides several novel features.

We consider an *unreliable* wireless network in that cells provide only *incomplete* spatial coverage and messages could be lost even within a cell due to localized *physical obstructions* or any other reason [5]. Mobile hosts roam unpredictably and they do not necessarily move among physically adjacent cells. A mobile host (MH) that leaves a cell may enter any other cell, or even re-enter the same cell. Furthermore, before re-entering, MH could remain unreachable for a "long" amount of time. The resulting scenario is quite general because it accommodates contemporary wireless LAN's, disconnected modes of operation [15], long-range movements, picocellular wireless networks in which the cell size is of the order of a few meters, such as a room in a building [11].

The protocol supports multicast communication within a group of MHs[1]. Communication is *reliable* in the sense that each group member delivers every multicast and discards duplicates. The sender of each multicast may select among three increasingly strong delivery ordering guarantees: *FIFO* (i.e., single-source ordering); *Causal* (i.e., deliveries occur in an order consistent with the "happened-before" ordering of transmissions [14,16,3]); *Total* (i.e., all group members deliver multicasts in the same order and this order is consistent with causal order). We are not aware of any multicast protocol for distributed mobile systems with

[1] Note that both senders and receivers are mobile hosts, unlike, for example, the proposal in [9], where senders are mobile but receivers are stationary.

these features. The protocol proposed in this paper is a generalization of a previous protocol that only provides *Total* ordering [5].

The protocol is such that movements of MHs do not trigger any message exchange in the wired network. In contrast, the protocol in [3] enforces causal order by broadcasting a message to *all* "mobile support stations" (MSS) upon *each* cell switching of *each* MH. Other protocols are based on the notion of *hand-off* [1,3,16], in which each cell switching triggers the movement of pending messages and administrative information from an MSS to another. In our protocol, data stored at MSSs improve performance but are not critical for correctness, i.e., MSSs merely *cache* state information whose primary copy is kept elsewhere in the wired network.

Correctness depends only on *global* assumptions about movements of users, i.e. of the form "a group member does not move very fast, *all the time*" [5]. Other protocols have more restrictive mobility assumptions: they assume that users do not move while certain actions are in progress [1] or (appear to assume) that the speed of users is always slower than the rate at which hand-off completes [16,3]. With our protocol, movements at inopportune times can only cause an occasional performance penalty and do not affect correctness. Furthermore, such penalty disappears when the mobile host stops moving "too fast".

Weak assumptions on the underlying computing system, absence of state information at MSSs and loose mobility assumptions, contribute to improve the reliability of applications deployed over the proposed protocol. Finally, the proposed protocol appears to be a good foundation for developing a multicast support able to tolerate crash failures in the wired network. We elaborate on this claim in Section 4.

2. System model

We consider *mobile hosts* and *stationary hosts* that communicate through message-passing. Mobile hosts (MHs) may move and communicate through wireless links. Stationary hosts (SHs) are connected to a wired network that provides reliable and FIFO-ordered communication. Some SHs, called mobile support stations or *gateways*, may communicate also through wireless links. Each gateway defines a spatially limited *cell* covered by a wireless link. A gateway may broadcast messages to all MHs in its cell and send messages to a specific MH in its cell, whereas a MH may only send messages to the gateway of the cell where it happens to be located. We do not assume any network support for routing messages to a specific MH. To take into account physical obstructions, communication within a cell is FIFO-ordered but messages may be lost. Furthermore, MHs may roam in areas that are not covered by any cell (Figure 1 top). Host crashes are beyond the scope of this paper (see also Section 4). SHs may communicate among themselves via FIFO-multicast, whose implementation is omitted as it does not provide any additional insight. Each gateway becomes aware of which MHs are in its cell by means of a *beaconing* protocol. The minimum requirements that we place on such a protocol are detailed in [5].

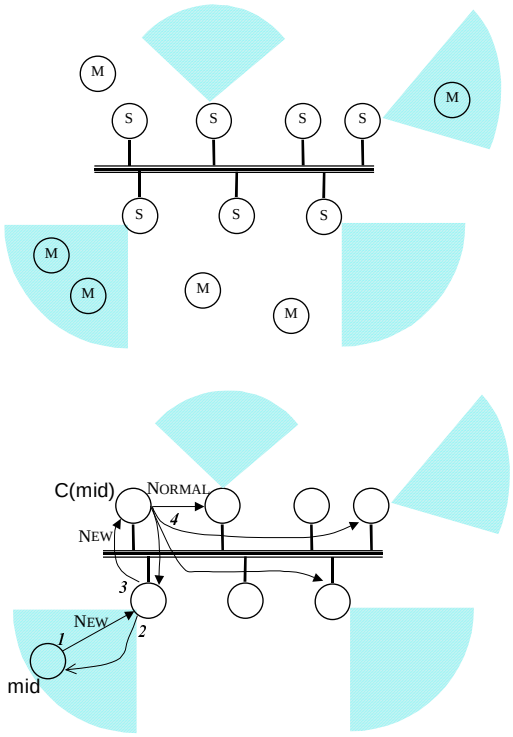

Figure 1: System with 7 SHs, 4 gateways, 6 MHs (top). Example of message-exchange pattern for F-Cast() and C-Cast()(bottom).

3. The protocol

3.1. Interface

The protocol supports multicast communication among members of a *group*. The protocol supports dynamic membership but, for space reasons, we shall present here only a simplified version that assumes a static membership. Handling of dynamic membership is detailed in [2]. Each group member may send multicasts to the other members, through the following primitives: `F-Cast()`, that provides FIFO-ordered delivery; `C-Cast()`, that provides causally-ordered delivery [14,16,3]; and `T-Cast()`, that provides a total order on multicast delivery that is consistent with the causal order of transmissions. More in detail, consider the following properties [12, 4]:

- (**FIFO Order**) If a group member sends a message m2 after sending a message m1, then any group member that delivers both messages delivers first m1 and then m2.
- (**Causal Order**) If the transmission of a message m2 causally follows the transmission of a message m1, then any group member that delivers both messages delivers first m1 and then m2.
- (**Total Order**) Any two group members that deliver a pair of messages m1 and m2, deliver them in the same order, i.e., either they both deliver m1 before m2, or they both deliver m2 before m1.

FIFO Order is guaranteed for any pair of messages. Causal Order is guaranteed for any pair of messages in which m2 is sent via either `C-Cast()` or `T-Cast()`. Total Order applies to any pair of multicasts sent via `T-Cast()`. Moreover, each multicast is delivered exactly once, i.e., duplicates are discarded.

Group members indeed deliver messages under reasonable assumptions on their physical movements. A group member M stops delivering messages if: (I) M starts entering and leaving cells so quickly that its messages never arrive to any gateway or messages from gateways are systematically lost; and (II) this pattern of movements persists forever. Our mobility assumptions can thus be informally summarized as "a group member cannot move very fast, all the time" [5]. Of course, if a group member M moves "very fast" for a certain time interval, then, during that interval, M could not be able to deliver messages. However, when M will slow down, it will start again delivering messages and will recover all lost messages. In other words, inopportune movements may affect only the *performance* of the protocol, not its *correctness* — i.e., when a mobile moves too fast it experiences high delivery latency, but messages are not lost and ordering guarantees are preserved. Other protocols are built upon mobility assumptions of a different form, e.g., a mobile computer does not switch between cells while state information about it is being redistributed in the static network (i.e., while hand-off is in progress) [1]. Similar assumptions may be critical for reliability: the correctness of a protocol should not rely on the assumption that users do not move while a certain part of the protocol is in progress. In our case users may move at any time, irrespective of what is going on at the protocol level, which favor reliability of applications deployed over the protocol.

3.2. Implementation overview

For ease of presentation we shall assume that all group members run on MHs and that there is at most one group member at each host. We say that a host C *receives* a message m when m arrives at the protocol at C. We say that C *delivers* m when the protocol forwards m up to the application. The first field of each message is a value of an enumerated type called *tag* and indicated in SMALLCAPS. We shall refer to fields of a message with the dot notation, e.g. m.f for field f of message m.

Each group member has a unique *member identifier*, denoted mid. A statically defined set of SHs, called *coordinators*, constitutes the *coordinator team*. A statically defined coordinator is the *team boss*, indicated CB. Each group member is associated with a coordinator C≠CB. As clarified below, each coordinator processes the multicasts issued by group members with which it is associated. `T-Cast()` are processed also by the team boss CB. The proposed protocol is a generalization of a previous protocol that supports only `T-Cast()` and makes use of a single coordinator [5].

A MH encloses its member identifier and a coordinator identifier in each message m that it sends. Upon receiving m, the gateway sends an acknowledgment ack(m) to MH and forwards m to the coordinator specified in m (forwarding does not occur when m is tagged NACK, see Section 3.3). MH periodically re-sends a copy of m until receiving a matching acknowledgment ack(m). If the MH roams in an uncovered region before receiving the acknowledgment, retransmissions are postponed until entering a cell again. An MH can move at *arbitrary* times. For instance, MH could send m while in a cell, leave the cell before receiving ack(m), re-send m in a new cell and so on, until receiving the expected ack(m) from a gateway.

We point out that, in our system model, retransmissions and acknowledgments are mandatory, due to the combination of: (I) incomplete spatial coverage, (II) unreliable communication within cells, (III) unpredictable movements of MHs. Of course, by removing one or more of these features, one could build a protocol where communication between MH and gateways does not need retransmissions and acknowledgments — e.g., the protocol in [1] assumes complete spatial coverage, no message loss within cells, absence of movements during hand-off. Clearly, retransmissions imply a cost in terms of power consumption and wireless bandwidth usage but, if one admits that messages can be lost, such cost can hardly be avoided.

A group member mid issuing either `F-Cast()` or `C-Cast()` sends a NEW message containing the payload to its own coordinator C(mid). mid maintains a Sent sequence number that increases upon sending each NEW message. This sequence number is enclosed in the NEW message, which enables C(mid) to process NEW messages in the same order in which they were sent and to discard duplicates (retransmissions and unpredictable

movements of MHs make it possible the delivery of out-of-order and duplicate messages at coordinators). C(mid) transforms the NEW message into a NORMAL message, assigns a sequence number cseq and multicasts the resulting message to gateways (Figure 1 bottom). Gateways then broadcast NORMAL messages to group members in their cells. Handling of NORMAL messages at group members is detailed in the next section. If mid issues a `T-Cast()`, then C(mid) constructs a NORMAL message as above but does not multicast it to gateways. Instead, C(mid) forwards the NORMAL message to the team boss CB, that appends a further sequence number cseqB and multicasts the resulting message to gateways.

The structure of a NEW message is <NEW, mid, Coord, Order, Sent, Deliver, p>, where: mid identifies the sending group member; Coord identifies its coordinator C(mid); Order specifies the multicast primitive invoked; Sent is the sequence number selected by mid; Deliver is an array with one entry per coordinator, where Deliver[k] contains the largest sequence number among the messages delivered by mid and sent by Ck at the moment mid sends this message; p is the payload. As an optimization, when Order specifies `F-Cast()` the array Deliver may be omitted.

The structure of a NORMAL message is identical to that of the matching NEW message except for the tag and for the following additional fields inserted by C(mid): cseq, a sequence number maintained by C(mid) and increased upon sending each NORMAL message; cseq-mid, the value of cseq assigned by C(mid) upon processing the last NEW message sent by mid (*originated* by mid for short). When the NORMAL message is processed by CB, CB adds a further field cseqB. It is a sequence number that CB increases upon sending each NORMAL message. In the following, the (unqualified) sequence number of a NORMAL message m identifies field m.cseqB if m is a `T-Cast()` and field m.cseq otherwise.

3.3 Handling of NORMAL messages at group members

Each group member mid maintains the following data structures:

- `mystable`: a table with one entry per coordinator. The entry associated with Ck, denoted `mystable[Ck]`, is a sequence number with this meaning: mid has delivered all messages sent by Ck and having sequence number $s \leq$ `mystable[Ck]`.
- `myahead`: a table with one entry per coordinator. The entry associated with Ck, denoted `myahead[Ck]`, is a *set* of sequence numbers with this meaning: mid has delivered all messages sent by Ck with sequence numbers in `myahead[Ck]`; furthermore, all elements in `myahead[Ck]` are greater than `mystable[Ck]`.
- `mybuf`: a memory region where to store received messages that cannot yet be delivered. Each MH may decide autonomously how much memory to allocate for mybuf. This array improves the performance of the protocol but it is not necessary for correctness. A received message m that cannot be buffered for lack of buffer space can be discarded (if there is no space because mybuf is full, one could even discard a buffered message to make room for m).

Initially, all entries of `mystable` are zero, all entries of `myahead` are the empty set, `mybuf` is empty.

A group member handles a received NORMAL message m by procedure HandleNormal() in Figure 2. Function Discardable(m) (lines 2 and 12) returns true iff m has already been delivered by the executing group member. Boolean function Deliverable(m) (line 3) returns true iff m can be delivered. The body of this function, not shown for brevity, evaluates the pertinent *delivery condition* for m, as clarified in the following. Consider predicates Delivered(C, s, mid) and DeliveredAll(C, s, mid), where C is a coordinator and s is the sequence number of a NORMAL message sent by C. Delivered(C, s, mid) is defined to be true iff the group member mid has delivered the message sent by C with sequence number s. DeliveredAll(C, s, mid) is true iff mid has delivered *all* messages sent by C with sequence number smaller than, or equal to, s. Function Deliverable(m) evaluates one of the following delivery conditions, selected on the basis of m.Order (my-mid indicates the executing group member):

`F-Cast()`:Delivered(m.Coord, m.cseq-mid, my-mid)
`C-Cast()`:Delivered(m.Coord, m.cseq-mid, my-mid) and (∀Ci, DeliveredAll(Ci, m.Deliver[Ci], my-mid))
`T-Cast()`: Delivered(m.Coord, m.cseq-mid, my-mid) and Delivered(CB, m.cseqB - 1, my-mid) and (∀Ci, DeliveredAll(Ci, m.Deliver[Ci], my-mid))

Translating the above delivery conditions in terms of mystable and myahead is obvious and we omit the related details for brevity. A message that is not a duplicate but does not satisfy its delivery condition may be buffered (lines 8 and 34). The delivery of a message may make it possible that the delivery conditions of some buffered messages evaluate to true (lines 6 and 25). Function ScanBuffer() (not shown) scans mybuf starting from the first message inserted and returns the first deliverable message found. When such message cannot be found, ScanBuffer() returns null.

When a group member buffers a received message, it requests retransmission of the missing messages immediately (line 9, the body of procedure AskRetransmission() is not shown.). The rationale is the following: since the wired network and the wireless network preserve FIFO order, the missing messages have been probably, although not certainly, lost. Modifying the protocol so that retransmission is requested only "after a while", however, is straightforward. Retransmission is requested by sending to the gateway a NACK message m: <NACK, mid, pair1,…, pairN>, where there is a pair field for each coordinator. The field associated with Ck consists of two sequence numbers $s1_k$ and $s2_k$ with this meaning: m.mid has delivered all messages sent by Ck with sequence number up to $s1_k$ (included) but it has missed those with sequence number $s \in [s1_k + 1,\ s2_k - 1]$; in case $s2_k = 0$, m.mid has missed all messages with sequence number $s \geq s1_k$.

Upon receiving a NACK message m, the gateway will relay to m.mid a copy of the missing multicasts through a sequence of TRANSFER messages. The gateway obtains a copy of the missing multicasts as clarified below. TRANSFER messages have the same structure as NORMAL ones except for the tag, that is different only for clarity. Group members handle TRANSFER messages in the same way as NORMAL messages. If the gateway detects through beaconing that m.mid has left its cell, then the gateway aborts the sequence of TRANSFER messages.

When mid constructs a NACK, it determines the values for the pair fields as follows: $s1_k$ = `mystable[Ck]`. Let $lowahead_k$ denote the smallest sequence number among those in `myahead[Ck]`, or 0 if such entry is empty; let $lowbuf_k$ denote the smallest sequence number among messages in `mybuf` sent by Ck, or 0 if there is no such message:

$s2_k = min(lowbuf_k, lowahead_k)$;
if ($s2_k$ == 0) then $s2_k = lowbuf_k$;
if ($s2_k$ == 0) then $s2_k = lowahead_k$;

Clearly, mid shall not send a NACK each time it buffers a message, because it could have just sent one. On the other hand, the fact that mid has sent a NACK does not imply that it will receive all required multicasts, because some TRANSFER messages could be lost. For this reason: (I) upon sending a NACK, mid sets a timer; (II) while the timer is set, mid does not send any further NACK; (III) when the timer expires, mid sends a further NACK only if it is still aware that it missed a multicast (i.e., there are still buffered messages).

Gateways maintain locally a cache of previous NORMAL messages that use for speeding up the processing of NACK messages. Whenever a gateway experiences a cache miss while processing a NACK, it will contact the proper coordinator. Each coordinator stores a copy of each NORMAL message m previously sent until it knows that m is *stable*, which means that m has been delivered by all group members [8]. Coordinators acquire stability information as follows. When a gateway receives a NACK message m, it extracts from each $m.pair_k$ field the sequence number $s1_k$ and sends to each coordinator Ck a message <STABINFO, m.mid, $m.s1_k$>. Upon receiving this message, Ck learns that m.mid has certainly delivered all messages with sequence number $s \leq m.s1_k$.

```
1  Procedure HandleNormal(m:message); {
2    if Discardable (m) then discard m;
3    elseif Deliverable(m) then {       // Not shown
4       deliver  m
5       UpdateData(m);
6       CheckBuffer();
7    } else {
8       Buffer(m);
9       AskRetransmission();            // Not shown
10   }
11 }
12 Function Discardable(m:message): boolean;
13 { return(m.cseq ≤ mystable[m.Coord] or
           m.cseq ∈ myahead[m.Coord]); }
14 Procedure UpdateData (m:message); {
15   if m.cseq = mystable[m.Coord] + 1 then {
16      mystable[m.Coord] = mystable[m.Coord] + 1;
17      while mystable[m.Coord]+1 ∈ myahead[m.Coord]
18      do {
19         remove mystable[m.Coord]+1
20               from myahead[m.Coord];
21         mystable[m.Coord] = mystable[m.Coord] + 1;
22      }
23   } else insert mystable[m.Coord] in myahead[m.Coord];
24 }
25 Procedure CheckBuffer();{
26   m1=ScanBuffer();                  // Not shown
27   while m1<>null do {
28      extract m1 from mybuf;
29      deliver m1;
30      UpdateData(m1);
31      m1=ScanBuffer();               // Not shown
32   }
33 }
34 Procedure Buffer(m);
35 { if there is space for m in mybuf and m is not in mybuf
36   then insert m in mybuf }
```

Figure 2: Main actions at group members

Consider a gateway G that is sending a sequence of TRANSFER messages originated by Ck to mid. If G receives from mid a further NACK message m, G

handles m as follows. Let [$slow_k$+1, $shigh_k$-1] be the sequence numbers that characterize the in-progress sequence and let $s1_k$, $s2_k$ denote the pair field of Ck in the just received NACK message m: (I) if $s1_k \leq slow_k$, then G discards m; otherwise, (II) G aborts the sequence and starts a new one according to $s1_k$, $s2_k$.

3.4. Discussion of delivery conditions

Condition Delivered(m.Coord, m.cseq-mid) obviously enforces FIFO Order among messages originated by m.mid. As an example, consider Figure 3 (top).

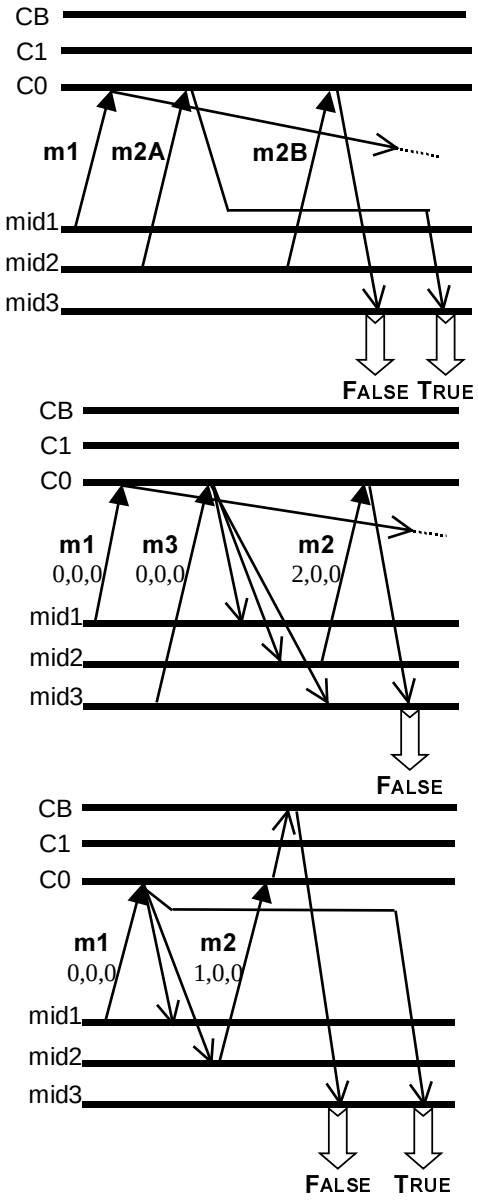

Figure 3: Examples for clarifying delivery conditions (see the text).

Messages going up are tagged NEW, those going down are tagged NORMAL and some NORMAL messages are omitted for clarity. Intermediate message-exchange at gateways are omitted for simplicity: mobility and incomplete spatial coverage manifest themselves as loss of FIFO order between messages from a coordinator to a MH (loss of messages is not considered in this figure). Suppose m1, m2A and m2B are all sent via F-Cast() and consider the handling of m1, m2A and m2B at mid3. Notice that the three messages are originated by different group members but these are associated with the same coordinator. mid3 buffers m2B because it has not yet received m2A (m2B.cseq-mid = *2*); upon receiving m2A, mid3 delivers m2A *and* m2B even though it has not yet received m1.

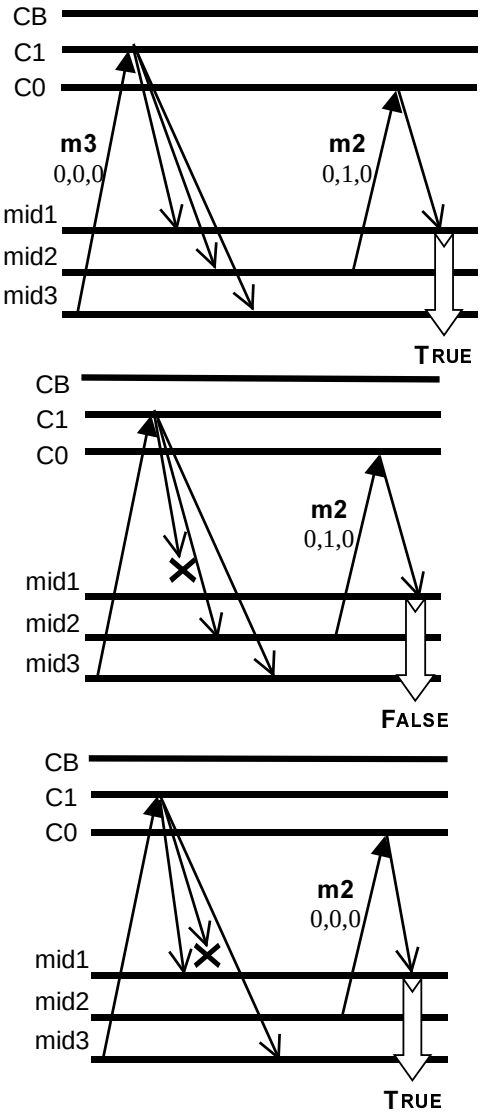

Figure 4: Examples of delivery conditions at mid1 upon receiving C-Cast(m2) issued by mid2.

Examples about the delivery condition of C-Cast(), are given in Figure 4. Numbers next to each message indicate the value of the Deliver array enclosed in the message. Entries in the array follow the order C0, C1, CB. The examples show the delivery condition at mid1 upon receiving m2, issued via C-Cast(), in three possible scenarios.

To motivate the delivery condition of C-Cast(), consider a group member mid that issues C-Cast(m) and suppose that mid does not issue any other multicast until delivering m. It is straightforward to see that the subcondition ∀Ci, DeliveredAll(Ci, m.Deliver[Ci]) enforces a delivery order consistent with Causal Order: m cannot be delivered if any message delivered by mid at the time it originated m has not been delivered already. Suppose now that mid can issue further multicasts even before delivering m. In this case, one must be sure that any message m1 sent by mid after m be delivered after m. This is guaranteed by the sub-condition Delivered(m.Coord, m.cseq-mid), because all multicasts issued by mid are sequentialized by the same coordinator C(mid).

The resulting delivery order is stronger than Causal Order: it could order pairs of causally unrelated messages sent by group members that happen to have the same coordinator. It could happen that a MH buffers a message m1 and waits for another message m2 that, actually, need not be delivered before m1. As an example, suppose m2 in Figure 3 (middle) be sent via C-Cast(). Values of cseq attached by C0 to m1, m3, m2 are 1, 2, 3, respectively. When mid3 receives m2, it does not realize that it could deliver m2 immediately: the delivery condition does not hold because mid3 has not yet delivered *all* messages of C0 with sequence number cseq ≤ 2; however, message m1 with cseq = *1* does not causally precede m2.

The ultimate reason for this feature (called *inhibition* in [3]) is because causality information is maintained on a per-coordinator basis, rather than per-group member. The advantage is that the size of data structures at MHs and message size overhead depend on the number of coordinators, not on the number of group members. Notice that an unnecessary buffering of a message m2 sent by C may occur only when: (I) m2 is "overtaken" by another message m1 sent by the *same* coordinator C; and (II) m1 and m2 were originated by *different* group members (Figure 3 middle). Since the underlying network is FIFO, this is going to happen only if the receiving group member misses m2 and receives m1, for instance, because of its movement across cells. Furthermore, as shown by results in [3], occasional unnecessary buffering does not necessarily result in overall worse performance.

The delivery order is intermediate between two extremes: with only one coordinator C-Cast() would provide Total Order, whereas with one coordinator per group member C-Cast() would provide a "pure" Causal Order, that is, it would not force any delivery order among causally unrelated pairs of messages. It follows that tuning the number of coordinators allows balancing between latency and memory/message size overhead.

Finally, as for T-Cast(), the sub-condition Delivered(CB, m.cseqB-1) enforces in-order delivery of messages sent by CB. This sub-condition alone would enforce a Total Order that might not be consistent with Causal Order. As an example, see Figure 3 (bottom). Suppose that m1 be sent with F-Cast() and m2 with T-Cast(). When mid3 receives m2, Total Order alone would allow delivering m2 immediately, but in that case Causal Order would not be satisfied (i.e., the delivery condition must be false, as indicated). To achieve Causal Order it suffices to couple the sub-condition being discussed with the condition for C-Cast().

Intuitively, the intermediate processing step at C(mid) is necessary to enclose cseq-mid in the message, which allows preserving causal order without maintaining per-group member counters.

4. Discussion

When a MH switches between cells, or disconnects, SHs need not exchange any state information about that MH. That is, when a MH moves, *no* message need be generated on the wired network.

Location information need not reflect "instantaneously" the actual cell compositions and may be (temporarily) inaccurate: Gateways use the beaconing protocol only for detecting whether a given sequence of TRANSFER messages should be aborted (i.e., the receiving MH has left the cell). In other words, we do not require that the beaconing protocol be able to track "perfectly" all moves. For example, a MH could leave and re-enter a given cell very quickly. Even if this movement were not tracked by the beaconing protocol, the multicast protocol would still be correct: the movement might cause some message loss, but these can be recovered automatically, as if the MH had moved behind a physical obstruction within the cell. From another point of view, we do not require that location information be consistent across the network. For example, two gateways could temporarily believe that a given MH belong to the respective cell because executions of the beaconing protocol at different gateways proceed autonomously from each other. This temporarily inconsistent location information, however, is harmless.

Although we have assumed that coordinators multicast NORMAL messages to all gateways, it may be

useful to send such messages only to those gateways whose cells indeed contain group members. The simple technique presented in [5] can be used to this purpose. Essentially, each gateway should just notify coordinators when its cell switches between empty and non-empty, in order to make it explicit its own interest in NORMAL messages. Group members could experience a slight increase in latency whenever they enter an empty cell, but this is a price to pay for supporting more efficiently systems where the number of cells may be much larger than the number of group members. Providing a similar functionality in multicast protocols that use hand-off is more costly and more complicated, because it may require updating a data structure (i.e., the list of gateways whose cell is non empty) replicated at *all* gateways whose cell is non empty [1]. Such updates, that must be serialized, are necessary whenever a cell switches between empty and non empty.

Buffering space at mobile computers is not necessary for *correctness*, but it is merely an *optimization*. Upon receiving a message m that should be buffered, one could even build the NACK and then discard m. Enclosing this idea in the protocol is fairly simple [5]. Each MH may thus trade buffering space against likelihood of unnecessary NACK messages, which may be useful for MHs with scarce memory resources.

The fact that stability information is piggybacked into other protocol messages that group members have to send anyway (i.e., NACK) is beneficial for reducing power consumption on the MHs and for decreasing the aggregate load on the wireless links. Other protocols instead acknowledge every multicast individually [1,3]. Of course, if a given group member never misses any message then no stability information about that member will be ever acquired. The key aspect is that collection of stability information may proceed asynchronously with the "ordinary" message exchange of the protocol, thus one may devise a number of ways for propagating stability information if necessary (e.g., a group member could piggyback stability information about itself into NEW messages, or a coordinator could solicit stability information through a dedicated field into NORMAL messages).

The protocol makes use of a statically defined set of coordinators in the wired network. The size of message headers and data structures at mobile hosts depends only on the number of coordinators rather than on the number of group members, which is useful for saving scarce resources on mobile hosts [3]. On the other hand, the latency of delivery may increase as the number of coordinators decreases (i.e., each coordinator has to process more multicasts). An implementation may thus allocate on the wired network the set of coordinators that provides a trade-off suitable to its own needs.

Messages sent via T-Cast()have to be processed by the team boss CB in addition to the coordinator associated with the sending group member. The coordinator CB might constitute a potential bottleneck when the number of messages requiring Total ordering becomes very large. A similar problem might arise, irrespective of the primitive used to send messages, when there is a single coordinator for all group members. However, by simulation we found that the likely bottleneck of the system, rather than the coordinator, appears to be the wireless interface at *gateways* [2].

We considered a scenario with one coordinator (apart from CB) and characterized by the following parameters: wireless bandwidth 10 Mbps, wired bandwidth 100 Mbps[2], several message rates up to an aggregate message rate of 2000 512-bytes messages/sec (corresponding to an aggregate bit rate of approximately 8.2 Mbps[3]). The wireless coverage was assumed to be complete and the message loss probability was set to 0.001 (message losses are assumed as independent). The simulation results show that the average total delay experienced by messages at the coordinator is very small when compared to the average total delay experienced at the gateway. For example, with an aggregate message rate 2000 512-byte messages/sec (aggregate bit rate 8.2 Mbps), we found an average latency 6.03 ms, with average delay at gateways and coordinator 2.21 ms and 0.07 ms, respectively. The total delay experienced at either a coordinator or a gateway, has three components: queuing, transmission and processing. Mainly due to the mismatch between the bandwidth of the wired and wireless networks, the queuing component of the delay experienced at the gateway tends to become the predominant factor of the overall latency and increases as the aggregate message rate increases approaching the wireless network's capacity (see [2] for details).

We argue that even the limitation due to the wireless interface between gateways and group members is not a peculiarity of our protocol. Instead, it is a consequence of the coexistence of different network technologies with significantly different bandwidth. It follows that any multicast protocol based on a "two-tier" architecture like ours (i.e., in which MHs offload storage, processing and transmission load to SHs) [1], is likely to exhibit a bottleneck at gateways. The observation that gateways are likely to constitute an intrinsic potential bottleneck

[2] 100 Mbps is the bandwidth currently provided by Fast Ethernet. The value of 10 Mbps for the wireless bandwidth was intentionally chosen greater than the bandwidth provided by current Wireless LANs (in the order of 1 Mbps) to banish the doubt that the single coordinator may become the critical point when the wireless bandwidth increases.

[3] The actual aggregate bit rate on the wireless network is even greater since it includes traffic caused by ACK, NACK, TRANSFER and retransmitted NEW messages

is particularly useful, because it allows using a simple, logically centralized coordinator-based protocol. As an aside, we found that the number of messages discarded by group members as duplicate messages is very small (less than 1 every 1000). Hand-off would prevent the arrival of duplicates at group members, but this result shows that insisting on preventing duplicates is probably not worthwhile.

We conclude this discussion by observing that the proposed protocol can be extended fairly easily for tolerating crash failures of SHs. We elaborate on this claim in the following. Crashes of gateways will affect only *availability*, e.g., if the gateway of a given cell is crashed, then communication within that cell will not be possible and the related spatial region will be part of the uncovered area. *Correctness* will not be affected because gateways do not store any state information about MHs: they only *cache* such information for decreased latency (i.e., copies of past NORMAL messages). Crashes of coordinators could be tolerated by means of systematic and established techniques for fault-tolerance, e.g., replication. The role of each single coordinator should be played by a set of SHs (perhaps the same set for all coordinators). By properly structuring interaction among these SHs, one could implement a "coordinator service" that is available to its clients (MHs and gateways) in spite of crashes of one or more of its components[4]. For example, tools that simplify the programming of such interactions are primary-backup replication and virtual synchrony [6,7].

Multicast protocols based on the notion of hand-off are much more difficult to make resilient to crashes of SHs. Not only gateways store state information (i.e., lists of messages directed to a given MH but not yet delivered), they also have to cooperate among themselves upon *each* movement of MHs and they have to maintain location information that is fully consistent across the network [1]. For example, a gateway could crash in the middle of an hand-off, making its state information unavailable; to tolerate this, one could replicate the information stored at that gateway on other SHs; but then, one would have to run a replica control protocol upon *each* movement of group members (and such protocol itself would have to be made resilient to crashes).

5. Concluding remarks

We have presented a multicast protocol for mobile systems that provides a reliable delivery service (delivery of every multicast and absence of duplicates) and accommodates neatly three increasingly strong delivery ordering guarantees. The protocol makes loose assumptions on the underlying computing system and on the user mobility pattern. Mobility of users need not trigger the exchange of any state information in the wired network. MHs may have scarce resources and, in particular, limited buffering space. It is possible to tune the amount of resources on the wired network in order to achieve the desired trade-off between run-time costs, i.e., message size and memory overhead against latency. A coordinator-based architecture makes the protocol simple and amenable to extension toward tolerance of crashes of SHs. Simulation results show that coordinators are not potential bottlenecks. We are performing simulation studies in order to evaluate in detail the performance of the protocol, in particular, with respect to group member mobility, probability and length of disconnection, ratio between number of group members and number of coordinators.

[4] The main problem will be that a message sent by an MH could arrive at a gateway and then get lost because of a crash of that gateway. This problem can be solved by requiring that the message will be acknowledged only when it has been processed by the coordinator service rather than by the gateway itself.

Acknowledgments

The work described in this paper has been carried out under the financial support of the Italian Ministero dell'Università e della Ricerca Scientifica e Tecnologica (MURST) in the framework of the MOSAICO (Design Methodologies and Tools of High Performance Systems for Distributed Applications) Project.

References

[1] A. Acharya, B. Badrinath, “A framework for delivering multicast messages in networks with mobile hosts”, *ACM/Baltzer Mobile Networks and Applications*, vol.1(2), June 1996, pp. 199-219.

[2] G. Anastasi, A. Bartoli, F. Spadoni, “A flexible multicast protocol for distributed mobile systems”, Technical Report PI-DII/1/99, February 1999.

[3] S. Alagar, S. Venkatesan, "Causal Ordering in Distributed Mobile Systems", *IEEE Transactions on Computers*, 46 (3), March 1997, pp. 353-361.

[4] M. Ahuja, M. Raynal, "An Implementation of Global Flush Primitives Using Counters", *Parallel Processing Letters*, vol. 5 (2), 1995, pp. 171-178.

[5] A. Bartoli, "Group-Based Multicast and Dynamic Membership in Wireless Networks with Incomplete Spatial Coverage", *ACM/Baltzer Mobile Networks and Applications*, vol.3(2), June 1998, pp. 175-188.

[6] K. Birman, “The process group approach to reliable distributed computing”, *Communications of the ACM*, vol.36, n.12, pp.36-53, December 1993.

[7] Ö. Babaoglu, A. Bartoli, G. Dini, “Enriched view synchrony: a programming paradigm for partitionable asynchronous distributed systems”, *IEEE Transactions on Computers*, vol.46, n.6, pp.642-658, June 1997.

[8] K. Birman, A. Schiper, P. Stephenson, “Lightweight causal and atomic group multicast”, *ACM Transactions*

on Computer Systems, vol.9, n.3, pp.272-314, August 1991.

[9] K. Cho, K. Birman, “A Group Communication Approach for Mobile Computing Mobile Channel: an ISIS Tool for Mobile Services”, Cornell University Technical Report TR94-1424, May 1994.

[10] G.H. Forman, J. Zahorjan, “The challenges of mobile computing”, *IEEE Computer*, vol.27, n.4, pp.38-47, April 1994.

[11] R. Ghai, S. Singh, “An architecture and communication protocol for picocellular networks”, *IEEE Personal Communications*, Third Quarter 1994, pp.36-46.

[12] V. Hadzilacos, S. Toueg, “Fault-tolerant broadcasts and related problems”, in *Distributed Systems (2nd edition)*, Sape Mullender ed., ACM Press 1993.

[13] F. Kaashoek, A. Tanenbaum, “An evaluation of the Amoeba group communication system”, *Proc. of the 16-th IEEE International Conference on Distributed Computing Systems*, May 1996, pp.436-447.

[14] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System", *Communications of the ACM*, vol.21 (7) , July 1978, pp. 558--565.

[15] S. Maffeis, W. Bischofberger, K.U. Mätzel, “A generic multicast transport service to support disconnected operation”, *ACM/Baltzer Wireless Network*, vol.2, n.1, January 1996, pp.87-96.

[16] R. Prakash, M. Raynal, M. Singhal "An Efficient Causal Ordering Algorithm for Mobile Computing Environments", *Proc. of the 16th International Conference on Distributed Computing Systems*, May 1996, pp. 744-751.