A Reliable Multicast Protocol for Distributed Mobile Systems: Design and Evaluation (extended version)

Giuseppe Anastasi, Alberto Bartoli*, Francesco Spadoni

Dipartimento di Ingegneria dell'Informazione, Università di Pisa, Via Diotisalvi 2, 56126 Pisa, Italy
Fax: +39-050-568522, E-mail: anastasi@iet.unipi.it

** Dipartimento di Elettrotecnica, Elettronica ed Informatica, Università di Trieste, Via Valerio 10, 34100 Trieste, Italy*
Fax: +39-040-6763460, E-mail: bartolia@univ.trieste.it

Abstract *Reliable multicast is a powerful communication primitive for structuring distributed programs in which multiple processes must closely cooperate together. In this paper we propose a protocol for supporting reliable multicast in a distributed system that includes mobile hosts and evaluate the performance of our proposal through simulation. We consider a scenario in which mobile hosts communicate with a wired infrastructure by means of wireless technology. Our proposal provides several novel features. The sender of each multicast may select among three increasingly strong delivery ordering guarantees: FIFO, Causal, Total. Movements do not trigger the transmission of any message in the wired network as no notion of hand-off is used. The set of senders and receivers (group) may be dynamic. Size of data structures at mobile hosts, size of message headers, number of messages in the wired network for each multicast, are all independent on the number of group members. The wireless network is assumed to provide only incomplete spatial coverage and message losses could occur even within cells. Movements are not negotiated and a mobile host that leaves a cell may enter any other cell, perhaps after a potentially long disconnection. The simulation results show that the proposed protocol has good performance and good scalability properties.*

1 Introduction

Recent technological trends have greatly stimulated distributed systems research and practice towards the support for *mobile* hosts. Laptops, cellular telephones, wireless networks, palmtop computers, wearable devices, allow users to carry computers to where their presence is actually needed. Not only this functionality may greatly extend the scope of existing applications of distributed systems, it may also enable novel application domains in such fields as inventory control, factory automation, on-site data collection, traffic monitoring and so on.

In this paper we propose a protocol for *reliable multicast* communication within a group that may include mobile hosts. By reliable multicast we mean, very informally, that all multicasts are delivered and there are no duplicates. Reliable multicast is a communication primitive that has proven its utility in the context of stationary distributed computing, in particular, where the application requirements imply a tight cooperation among a number of remote entities that must maintain some form of shared state. Our protocol extends the applications of this paradigm towards distributed mobile systems.

Mobile hosts typically need a special treatment, for a number of reasons. Traditional network protocols implicitly assume that hosts do not change their physical location over time **[BPT96]**. Mobile devices have severe resource constraints in terms of power consumption **[FZ94]**. Processing and storage resources of portable devices are much more limited than those of stationary devices **[MDC93]**. Wireless bandwidth may be much smaller than wired bandwidth. Wireless communication tends to exhibit high error rates **[ES96]**. The energy cost of wireless communication is highly asymmetric, i.e. transmitting a message is much more costly than receiving a message, up to one order of magnitude. Furthermore, mobility introduces new issues at the algorithmic level. For example, a mobile host may miss messages simply because of its movements, even with perfectly reliable communication links, with computers that never crash and with a full "flooding" of the network **[AB96]**. All the above reasons imply that specialized protocols are required for extending to mobile hosts functionalities that are common for stationary ones.

Our scenario consists of a distributed system in which mobile hosts (MHs) communicate with a wired infrastructure through a number of spatially limited cells that define wireless links. Cells provide only *incomplete* spatial coverage and wireless communication is *unreliable*. Movements are unpredictable in the sense that a MH may leave a cell without prior negotiation and then it may re-enter *any* other cell, perhaps after remaining out of coverage for some time. We do not assume any support for routing messages to a MH, i.e., we do not rely on Mobile IP **[P96]**. The resulting scenario is quite general because it accommodates contemporary wireless LAN's, infra-red networks requiring line-of-sight connectivity, physical obstructions, picocellular wireless networks where cells coincide with rooms in a building **[GS94]**, disconnected modes of operation, long-range movements.

The proposed protocol has the following salient features:

- Multicast communication is reliable, i.e., all multicasts are delivered and there are no duplicates.

- The sender of each multicast may select among three increasingly strong delivery ordering guarantees **[HT93]**: *FIFO* (i.e., single-source ordering); *Causal* (i.e., deliveries occur in an order consistent with the "happened-before" ordering of transmissions **[L78,PRS96,AV97]**); *Total* (i.e., all group members deliver multicasts in the same order and this order is consistent with causal order).
- The set of senders and receivers (*group*) may be *dynamic*: a mobile host may join and leave such group at will of the application.
- Cell switchings of MHs do not trigger any message exchange in the wired network. This feature allows supporting efficiently a large number of MHs that frequently switch between cells. The ability to accommodate neatly such a scenario is gaining importance due to the current trend toward smaller cells, which is motivated by their advantages in terms of improved aggregate throughput and smaller power required for transmitting **[CP98]**.
- Size of message headers, size of data structures at MHs, number of messages in the wired network for each multicast do not depend on the number of group members. These factors contribute to make the protocol highly scalable.

To the best of our knowledge, previous reliable multicast protocols for distributed mobile systems offered at most two of the following features, supported by our proposal: (i) incomplete spatial coverage; (ii) unreliable wireless communication; (iii) dynamic membership. None such protocol was able to accommodate cell switching without any message exchange in the wired network and none supported the three delivery guarantees FIFO, Causal, Total.

We have analyzed in detail the performance of the proposed protocol by simulation. We anticipate that the protocol has good performance and good scalability properties. For example, we shall see that the delivery latency with a fixed number of senders remains approximately constant up to a large number of receivers and that this latency is limited only by queuing times at "Mobile Support Stations", a factor that follows from the mismatch between the wired and wireless bandwidth and thus it is not a peculiarity of our protocol.

2 Overview and related work

In our protocol, new multicasts originated by MHs[1] have to be processed by a small subset of stationary hosts (SHs), the *coordinators*, that attach proper control information to these multicasts. The details of such processing and of the related control information depend on the delivery guarantee selected by the sender. Coordinators send the multicasts to *mobile support stations* (MSSs), i.e., SHs connected to the wired network and covering a cell each. MSSs broadcast messages from coordinators in the respective cell. MHs may miss multicasts, for any of the following reasons: (i) they roam out of coverage; or (ii) the wireless link looses the message; or (iii) they move at inopportune times, e.g., they switch cell before a multicast is broadcast in the old cell and after it has been broadcast in the new one. MHs detect missing multicasts and ask retransmissions to MSSs by means of a negative acknowledgment scheme. MSSs cache copies of past multicasts and, if a retransmission request cannot be satisfied with the cache content, they fetch the pertinent messages from coordinators, that store a copy of each past multicast that might have not been delivered by all group members. MHs piggyback acknowledgments for past multicasts into retransmission requests and MSS forwards such acknowledgments to coordinators, which enable coordinators to free storage resources as appropriate. Observe that MSSs do not interact between themselves upon cell switchings of MHS. The proposed protocol is an extension of a protocol previously developed by one of the authors that was based on a single coordinator and provided only Total ordering **[B98]**. A simplified version of the proposed protocol, without dynamic membership, has been presented in **[ABS99b]**.

We decided not to rely on network-level mechanisms for routing messages to a MH, e.g., Mobile IP, for several reasons. First, tracking the location of individual MHs is costly, both in terms of latency and of additional traffic on the wired network, as each cell switching requires updating the distributed routing information. Second, such a strategy would make it more difficult to exploit the broadcast capabilities of the wireless medium when many MHs belong to the same cell. Third, cell switchings would generate traffic in the wired network even while no new multicasts are generated. Fourth, we allow MHs to remain out of coverage for a time that is "long" if compared to the timeouts used at the transport level. Finally, the problem of recovering from missing messages due to inopportune cell switchings would remain. Multicast protocols that rely on Mobile IP are generally targeted at different application domains and provide unreliable, best-effort, unsequenced delivery **[XP97, CWBM98]**. In particular, no messages are delivered

[1] For simplicity and unless specified otherwise, we shall consider the term "mobile host" as synonymous of "group member".

during a cell switching and messages possibly lost will not be recovered in the new cell.

The protocol proposed in **[AB96]** supports *static* group membership and *FIFO-ordering* of deliveries (see also **[YHH97]** for clarifications about this ordering) and appears to assume *complete* spatial coverage. This protocol maintains on the wired network chains of pointers along which messages have to be forwarded to the current location of each group member. When a MH switches between cells, a dedicated *hand-off* procedure updates these pointers and moves the messages not yet delivered from the old MSS to the new one.

Our protocol does not maintain any chain of forwarding pointers in the wired network and, consequently, it does not employ any notion of hand-off. This feature is highly desirable for several reasons. First, hand-off generates traffic on the wired network even while no new multicasts are generated. Second, given the current trend toward smaller cells **[CP98]** one should privilege strategies that accommodate frequent cell switching efficiently. In particular, hand-off does not appear to be suitable for picocellular networks because of the large amount of information that need to be forwarded **[GS94]**. Third, the traffic related to mobility management tends to be high itself **[MAO92]**, thus one should strive to avoid further traffic for higher-level protocols.

Absence of hand-off is desirable also from other points of view that we briefly mention in the following. More details about these claims can be found in the cited papers. First, the resulting protocol may be made tolerant to crashes of SHs much more easily **[ABS99b]**. Essentially, hand-off implies that MSSs store critical state information and that this information has to travel across MSSs upon each cell switching, whereas in our approach MSSs merely *cache* state information and each MSS manages its cache *autonomously*, independently of cell switchings. Second, one can base correctness on more practical mobility assumptions. For example, **[AB96]** requires that MHs do not move while a certain portion of the protocol, related to hand-off, is in progress, and **[PRS96,AV97]** (appear to) require that the speed of users is always slower than the rate at which hand-off completes. In our case correctness does not depend on "when" a MH moves but it depends on *global* assumptions, i.e., of the form "a group member does not move very fast, *all the time*" **[B98]**. 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 MH stops moving "too fast".

The protocol in **[ARR97]** supports *FIFO-ordering* of deliveries within a *static* group. It is meant to reside on top of a *location directory*, i.e., a service that indicates the MSS currently associated with a given MH. It requires a form of hand-off whose details depend on the guarantees offered by the underlying location directory. Mobility thus generates traffic in the wired network even while MHs do *not* issue multicasts (the cited paper evaluates such overhead through simulation). The number of messages in the wired network per multicast grows with the number of receivers, e.g., around 5 and 7.5 with 2 and 5 receivers, respectively. Our protocol does not require a location directory, need not track the location of individual MHs, generates traffic on the wired network only when new multicasts are indeed generated, and imposes a load on the wired network that depends only on the number of MSSs, not on the number of MHs. With respect to such load, in the absence of message losses, FIFO and Causal ordering require a point-to-point message and a multicast to the mobile support stations, whereas Total ordering requires an additional point-to-point message. The protocol being discussed does not require that MHs store information about the multicasts received so far, whereas ours does. However, as will be clarified later, in our protocol such information is not necessary for *correctness* but it is merely an *optimization*, hence an actual implementation may trade memory resources against delivery latency according to its own needs. Furthermore, full performance can be obtained even by reserving just a few kilobytes, as we have observed through simulation.

The protocol in **[PRS96]** extends the proposal in **[AB96]** to preserve causal order of deliveries. State information that MSSs maintain on behalf of MHs, and that must be transferred upon hand-off, is augmented by a vector with one integer per MSS and a vector with one element per MH, where each element is a set of integer pairs. All arguments made about **[AB96]** apply to this protocol.

The protocol in **[AV97]** supports reliable causally-ordered message delivery in mobile distributed systems. This paper actually presents a family of three protocols, all derived from the protocol for stationary hosts in **[RST91]**, whose crucial differences include the size of the message headers. The first algorithm requires headers of $O(N_M^2)$, where N_M is the number of MHs, the second requires headers of $O(N_G^2)$, where N_G is the number of MSSs, the third requires headers of $O(k^2*N_G^2)$, where k is a positive integer. A variant to these protocols that require $O(N_M^2 * N_G^2)$ is presented in **[YHH97]**. Our protocol has message headers whose size depends only on the number of coordinators, that is typically much smaller than either N_M or N_G. The number of coordinators also determines the size of data structures at mobile hosts and may be selected as discussed in **[ABS99a]**. On other hand, the protocols in **[AV97,YHH97]** are not restricted to multicast

communication. As an aside, the protocols in **[AV97,YHH97]** are based on hand-off and, specifically, the protocol in **[AV97]** broadcasts a message to *all* MSSs upon *each* cell switching.

Reliable multicast protocols for mobile distributed systems generally insist on preventing the arrival of duplicates at MHs (the so-called *exactly-once* property) **[AB96,ARR97]**. Hand-off is the main tool for achieving this property. In our protocol a MH may occasionally receive duplicates of multicasts already delivered, due to the absence of hand-off and to the simple negative acknowledgment scheme used. Such duplicates will be discarded by the protocol layer at MH itself, i.e., without delivering them to the application.

We did not strive to achieve exactly-once on the ground that it does not really imply substantial battery savings: given the broadcast nature of the wireless medium, the lower protocol layers of *all* MHs in a cell have to handle each message directed to any of them — i.e., a MH has to handle many unnecessary messages anyway. Furthermore, as confirmed by our simulation results, duplicates do not consume wireless bandwidth significantly. Important battery savings could instead be obtained by minimizing *transmissions* operated by MHs. This feature is not exploited in **[AB96]**, where each multicast has to be explicitly and individually acknowledged by the MH. In contrast, we piggyback acknowledgments into other protocol messages that are needed anyway, i.e., for recovering from missing messages. Our simulations corroborate the intuition that messages for requiring retransmissions are typically much less than delivered messages.

The protocol in **[AB96]** is such that a MH has to acknowledge every received multicast before the MSS can send another multicast to it. This strategy consumes scarce wireless bandwidth and increases contention for the wireless link, thus contributing to increase the average end-to-end delivery latency. Our protocol does not need explicit acknowledgments and does not delay the transmission of new multicasts operated by MSSs. From this point of view, the protocol in **[ARR97]** takes an approach similar to ours.

3 System model

MHs and SHs communicate solely through message-passing. Communication in the wired network is reliable and FIFO-ordered. An MSS 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 MSS of the cell where it happens to be located. Communication within a cell is FIFO-ordered but messages may be lost. MHs may roam in areas not covered by any cell. Host crashes are beyond the scope of this paper.

Each MSS maintains a data structure, called local, that identifies the set of MHs in its cell. A MSS becomes aware of which MHs are in its cell by means of a *beaconing* protocol. Every T_B seconds, the MSS broadcasts a "beacon" within its cell. Upon receiving a beacon, a MH announces its presence in the cell by sending back a response "greeting" message that contains its host identifier. We shall omit the details of this protocol for sake of brevity and we shall only assume that it satisfies the following informal properties (we denote by $C(MSS_i)$ the cell associated with MSS_i): **(B1)** If a mobile host M enters a cell $C(MSS_i)$ and remains in $C(MSS_i)$ for a sufficiently long time, then M will be included in local at MSS_i; **(B2)** If a mobile computer M leaves $C(MSS_i)$ and remains outside of $C(MSSi)$ for a sufficiently long time, then M will be removed from local at MSS_i; **(B3)** Let $C(MSS_1),\ldots,C(MSS_n)$ be a set of cells that spatially overlap; a mobile computer that remains in the overlapping area for a sufficiently long time belongs to local of only one MSS in $MSS_1,\ldots,MSS_n$. The "sufficiently long time" is, essentially, T_B plus the time-out set by *MSS* for receiving the associated greetings. The actual value of T_B depends on the speed of user movements and on the cell size, we used 100 sec in our simulations. Property B3 may be enforced by mobile computers, for instance, on the basis of the "strength" of the beacon.

It may be useful to point out what follows:

- When a *MH* switches between cells, the related *MSSs* do not exchange any state information about this computer. They simply update autonomously the respective local.
- We allow messages exchanged between a *MH* and an *MSS* to be lost even while *MH* continues to belong to local at that *MSS*. For instance, there could be a physical obstruction in the related cell or *MH* could leave and re-enter this cell so quickly that its movement is not tracked by the beaconing protocol.
- A *MH* could (temporarily) belong to local at multiple *MSSs*. For instance, because the execution of the beaconing protocol triggered by a cell switching cannot complete simultaneously at both *MSSs* involved.

In short, location information need not reflect "instantaneously" the actual cell compositions and may be (temporarily) inaccurate. As we shall see, we interpret local only as a "hint" about the actual cell composition and use it only for allocation and deallocation of data structures at MSS. The content of local is never critical

for correctness.

4 The protocol

4.1 Interface

A MH becomes a group member by executing the Join() primitive and stops being a group member by executing the Leave() primitive (we consider only one group for ease of presentation). 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 **[L78,PRS96,AV97]**; 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 **[HT93]**:

- (**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, e.g. 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, e.g. duplicates are discarded.

The fact that group members indeed deliver messages is guaranteed under reasonable assumptions on their physical movements, i.e., "a group member does not move very fast, all the time" **[B98]**. In particular, a group member M stops delivering messages if: (I) M starts entering and leaving cells so quickly that its messages never arrive to any MSS or the matching acknowledgements are systematically lost; and (II) this pattern of movements persists forever. Similarly, a host wishing to be a group member indeed manages to become a group member unless it moves according to (I) and (II).

Some of the messages delivered by group members consist of *membership changes* that are triggered by Join() and Leave() primitives. Such messages are marked with a special flag and contain the current group membership. Property Total Order apply to membership changes.

4.2 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.

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. Initially, we shall assume that: (I) the group membership is static; (II) it is known to all coordinators; (III) each group member knows the identity of its own coordinator. We shall remove these assumptions at the end of this section.

A MH encloses its member identifier and a coordinator identifier in each message m that it sends. Upon receiving m, the MSS sends an acknowledgment ack(m) to MH and forwards m to the coordinator specified in m[2]. MH periodically re-sends a copy of m until receiving a matching acknowledgment ack(m). If MH roams in an uncovered region before receiving ack(m), retransmissions are postponed until entering a cell again. A 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 MSS.

A group member mid issuing either F-Cast() or C-Cast() sends a NEW message containing the payload to its own coordinator C(mid). C(mid) changes the tag to NORMAL, appends a sequence number and multicasts the resulting message to MSSs. MSSs then broadcast NORMAL messages to group members in the respective cells. A group member issuing a T-Cast() sends a NEW message as above and C(mid) constructs a NORMAL message as above. However, in this case, C(mid) forwards the NORMAL message to the team boss CB rather

[2] Messages tagged NACK do not include a coordinator identifier and MSS do not forward such messages, see Section 4.6.2.

than multicasting it to MSSs. The team boss appends a further sequence number and then multicasts the resulting message to MSSs.

Retransmissions and unpredictable movements of MHs make it possible the arrival of duplicate and out-of-order NEW messages at coordinators. Each group member mid maintains a sequence number that encloses in each NEW message. This sequence number enables C(mid) to discard duplicates and to process NEW messages from a given MH in the order in which they were generated rather than in the order in which they are received.

Incomplete coverage, unreliable wireless links, unpredictable movements of MHs, make it possible the loss of NORMAL messages at group members as well as the arrival of duplicate and out-of-order messages. A group member mid receiving a NORMAL message m determines whether: (i) m has to be discarded (i.e., it is a duplicate); (ii) m has to be delivered; (iii) m has to be buffered as its immediate delivery would violate the ordering specified by the sender. The details of such decision are given in section 4.6.3. Essentially, the ordering between NORMAL messages from a given coordinator is specified by the sequence number selected by that coordinator. The ordering between messages from different coordinators, necessary for preserving causal order, is specified by additional information enclosed by mid in the matching NEW message, i.e., the set of messages delivered at the moment it issued the NEW.

When a group member detects a hole in the sequence of received messages, it requests retransmission of the missing messages by sending a proper NACK to the MSS. A MSS that receives a NACK relays to the sending group member mid a copy of the missing multicasts, through a sequence of TRANSFER messages. TRANSFER messages are equivalent to NORMAL ones but are destined to a specific group member. MSS aborts the sequence if it detects through beaconing that mid has left its cell.

Each coordinator stores a copy of each NORMAL message m that it previously sent until it knows that m is *stable*, i.e., that m been delivered by all group members **[BSS91]**. It follows that a MSS can certainly find a copy of any missing multicast at the pertinent coordinator. For efficiency, however, MSSs maintain locally a cache of previous multicasts. In other words, if a group member mid remains unreachable "for a while" and then enters a cell, the MSS will use its local cache to bring mid up-to-date and, in case of "long" disconnections, it will perhaps fetch "old" multicasts from coordinators with FETCH messages.

Group members piggyback acknowledgments for multicasts already delivered into NACK messages. MSSs extract this information and forward it to coordinators within STABINFO messages. A coordinator thus acquires stability information about a group member whenever that member discovers that it missed a multicast.

Tag	From	To	Use
JOIN	MH	CB	For joining the group
NEW	MH	C	For multicasting in the group
NACK	MH	MSS	Notify about missing multicasts
ACK	MSS	MH	Make sure that a message has arrived at MSS
NORMAL	C or CB	MH	Propagate a multicast or a membership change
TRANSFER	MSS	MH	Recover from missing multicasts
FETCH	MSS	C	Retrieve missing multicasts
STABINFO	MSS	C	Propagate stability information
JOINSYNC	CB	C	Allocate data structures at coordinators
JOINOK	C or CB	CB	Communicate initial sequence number
LEAVESYNC	CB	C	Deallocate data structures at coordinators
LEAVEOK	C or CB	CB	Synchronization

Figure 1 Messages used in protocol.

Group joining and leaving occurs as follows. Initially, there are no group members. A MH wishing to become a group member sends to the boss CB a JOIN message[3]. Membership changes are propagated by

[3] The problem of determining the identity of CB is irrelevant to our discussion.

means of dedicated and totally-ordered NORMAL messages[4]. The joining MH learns from this message the identity of its own coordinator and the current sequence number of NORMAL messages for each coordinator — MH will deliver messages with greater sequence numbers. A group member wishing to leave the group sends a NEW message with a special indication. Upon receiving the matching acknowledgment from a MSS, the group member stops participating in the protocol. Retransmissions and unpredictable movements allow delivery of duplicates and out-of-order JOIN, much like NEW messages. Their handling is detailed in 4.6.1.

The messages exchanged in the protocol are summarized in Figure 1. Their structure is detailed in section 4.5.

4.3 Observations

Before proceeding further, we make the following observations:

- MH retransmits every message until receiving the matching acknowledgment from a MSS. 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 MSSs does not need retransmissions and acknowledgments — e.g., the protocol in **[AB96]** 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.
- Beaconing enables an MSS to know the identifiers of MHs in its cell, but in practice not all MHs will be group members. To this end, each MSS maintains a data structure, called local-m, containing the subset of local with MHs that are actually group members. We shall assume that a dedicated field of a "greeting" message (the MH's response to a "beacon" message from the MSS) may contain information of the form "I am a group member and mid is my member identifier". MSSs use this field to update local-m, if necessary.
- The details required for implementing reliable FIFO-multicast in the wired network are irrelevant to our discussion and will thus be omitted. For example, one could add a layer above the (best-effort) IP multicast if available **[DC90]**.
- When the number of MSSs may be much larger than the number of group members sending a NORMAL message to all MSSs is clearly an unnecessary cost. Depending on the operating environment, it might be useful to involve in the protocol only those MSSs whose cells actually contain group members. The simple extension presented in section 4.8 can be used to this purpose.
- Stability information may flow to coordinators in a variety of ways, not only with the simple method described. For example, group members could piggyback acknowledgments also in NEW messages. Coordinators could even solicit somehow explicit acknowledgments every now and then, for example when a group member does not miss any message for a long time. The key issue is that acknowledgements and deliveries proceed asynchronously.

4.4 Data structures

The data structures are given in Figure 2 and explained in the following. The following types are used: MembIdT, CoordIdT, MSSIdT for identifiers of group member, coordinator, MSS, respectively; SeqnumT for sequence numbers; MsgT for messages. CTable indicates a table with one entry per coordinator, whereas GTable indicates a table with one entry per group member. Of course, the former is allocated statically, whereas the latter is initially empty and it is up to the protocol to allocate and deallocate entries, as appropriate. Initially, all sets are empty, all items of type SeqnumT are zero and each GTable is empty. Unless stated otherwise, such initialization occurs also upon allocating a GTable entry, e.g., a new entry for cseq-memb is initialized to 0. We use associative addressing for CTable and GTable, e.g., cseq-memb[mid] indicates the entry of cseq-memb associated with mid. For ease of exposition, we shall assume that each coordinator allocates a GTable entry for each group member. Modifying the protocol so that a coordinator stores information only about those members with which it is associated is straightforward.

Each coordinator C_k maintains the following data structures. Here, "multicast" is a shorthand for "NORMAL message sent by C_k". Members identifies group members. Cseq is the sequence number of the last multicast.

[4] Consecutive membership changes differ for at most one MH. Modifying the protocol so that CB attempts to collect multiple JOIN and LEAVE into a single NORMAL message is straightforward.

Cseq-memb[mid] contains the value of cseq when C_k processed the last NEW message sent by mid, or 0 if C_k has not processed any such message yet. Recv[mid] contains the sequence number of the last in-order NEW message that C_k has received from mid. Reorder-buffer[mid] buffers the (non-duplicate) out-of-order NEW messages sent by mid. First[mid] contains the lowest sequence number among the multicasts that mid must deliver. Stable[mid] has this interpretation: mid has certainly delivered all multicasts with sequence number $s \in$ [first[mid],stable[mid]]; stable[mid] is 0 if C_k does not know which multicasts have been delivered by mid. Norm-buffer contains an element for each multicast that is not known to be stable; each element contains a copy of the multicast and a set (called pending) of MembIdT, one for each group member that might not have delivered the multicast yet. If the multicast is to be ordered totally, then the entry is created only at CB (i.e., not at the coordinator of the group member that sent the matching NEW). Finally, subscribed contains the identifiers of all MSSs to which NORMAL messages are to be addressed. This set is initialized statically and is never modified (see also section 4.8).

```
// At coordinators:
members:        set of MembIdT;
cseq:           SeqnumT;
cseq-memb:      GTable of SeqnumT;
recv:           GTable of  SeqnumT;
reorder-buffer: GTable of set of MsgT;
first:          GTable of SeqnumT;
stable:         GTable of SeqnumT;
norm-buffer:    set of (MsgT, set of MembIdT);
subscribed:     set of MSSIdT;
// The following are only at the boss CB:
member-cache:   set of MembIdT;
coords:         GTable of CoordIdT;
```

```
// At MSSs:
local-m:        set of MembIdT;
cseq:           CTable of SeqnumT;
first:          CTable of GTable of SeqnumT;
norm-cache:     set of MsgT;

// At group members:
mystable:       CTable of SeqnumT;
myahead:        CTable of set of SeqnumT;
mybuf:          set of MsgT;
sent:           SeqnumT;
delivered:      CTable of SeqnumT;
```

Figure 2 Data structures at coordinators, MSSs and group members

The boss maintains, in addition, the following. Member-cache identify recent past group members. An entry for mid is inserted into member-cache as soon as mid has left the group and will be purged after a time sufficiently long to ensure that no messages related to mid are still in transit (assumptions of this kind are practically reasonable and are often necessary in distributed computing **[L78,HR94]**, similarly to the choice of join-id operated by MHs, see section 4.4), Coords[mid] identifies the coordinator associated with mid.

Each MSS maintains the following data structures. Local-m identifies the group members in the cell (XXX). First[C_k] is a table with the same meaning as at C_k. Norm-cache is a cache of past NORMAL messages. Cseq[C_k] is the sequence number of the last NORMAL message received from C_k.

Each group member mid maintains the following data structures. Mystable[C_k] has this meaning: mid has delivered all messages sent by C_k and having sequence number $s \leq$ mystable[C_k]. Myahead[C_k] has this meaning: mid has delivered all messages sent by C_k with sequence numbers in myahead[C_k]; furthermore, all elements in myahead[C_k] are greater than mystable[C_k]. Mybuf contains received messages that cannot yet be delivered otherwise the delivery guarantees might be violated. Each MH may decide autonomously how much memory to allocate for mybuf. This buffer improves the performance of the protocol but it is not necessary for correctness, i.e., a received message m that cannot be buffered for lack of space in mybuf can be discarded (if there is no space because mybuf is full, one could even discard a buffered message to make room for m). Sent is the sequence number of the last NEW message sent. Delivered[C_k] is the largest sequence number among the NORMAL messages delivered by the group member and sent by C_k.

4.5 Structure of messages

A JOIN message has these fields: CB, the identifier of the boss; M, the (unique) host identifier of the sending MH; join-id, a bit pattern selected by MH so that it is different from any other join-id that MH selected in the past (e.g., a 32-bit random pattern **[L93]**). The pair (MH, join-id) constitutes the member identifier of the group member, i.e., mid.

A NEW message has these fields: mid, the identifier of the sending group member; coord, the identifier of C(mid); order, that specifies the multicast primitive invoked; sent and delivered, the value of the analogous data structures at the moment mid constructed the message (see Section 4.4); leaving, a flag that is set only if the sending group member wishes to leave the group; p, the actual payload. As an optimization, when order specifies F-Cast(), the array delivered may be omitted.

A NORMAL message has the same fields as a NEW message[5], plus the following fields inserted by C(mid): cseq, the value of the analogous data structure at the moment C(mid) constructed the message; cseq-mid, the value of cseq-mid[mid] at that moment. If the NORMAL message is to be totally ordered, then the boss CB inserts an additional field cseqB, containing the actual value of cseq at CB. In the following, when we refer to the "sequence number" of a NORMAL message m we shall refer to field m.cseqB if m is a T-Cast() and to field m.cseq otherwise.

A NORMAL message carrying a membership change is totally ordered and contains an additional memb field inserted by CB. This field is made up of the following subfields: mid, the identifier of the group member that either is joining or leaving, as specified by a flag; if mid is joining, then there is a subfield C, the identifier of the coordinator assigned to mid, and N subfields $s_1, s_2, \ldots, s_N$, one for each coordinator. s_j ($j \in [1,N]$) is the smallest sequence number of a NORMAL message issued by C_j that mid must deliver.

A NACK message has these fields: mid, the sending group member; a $pair_k$ field for each coordinator C_k from which mid has missed messages. This field consists of two sequence numbers $s1_k$ and $s2_k$ with this meaning: mid has delivered all messages sent by C_k 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 $s2_k = 0$, mid has delivered no message with sequence number greater than $s1_k$.

A STABINFO message has these fields: mid, that identifies a group member and a sequence number s such that mid has certainly delivered all multicasts, issued by the coordinator to which the STABINFO is addressed, with sequence number up to s (included).

For brevity, we omit the structure of the remaining messages, that do not provide any additional insight.

4.6 Actions

In the next sections we shall describe the main actions performed at group members, MSSs and coordinators. For the sake of brevity, we shall present only those details that cannot be easily determined from section 4.2, section 4.5, section 4.4, Figure 2. Correctness proofs can be built similarly to **[B98]**.

4.6.1 Actions at coordinators

Upon receiving a JOIN, the boss CB multicasts a JOINSYNC message to all coordinators (including itself) and waits for a JOINOK from each of them. Then it selects a coordinator for mid according to some fair rule, records the association in coords and finally issues a NORMAL message whose memb field describes the joining of mid (this field is filled with information extracted from JOINOK messages, see also below).

Upon receiving a JOINSYNC identifying a joining group member mid, a coordinator C_k inserts mid in members, allocates an entry for mid in all GTable items and sets first[mid] equal to cseq. Finally it responds to CB with a JOINOK containing the value of first[mid].

Upon receiving a NORMAL message with the leaving flag set, the boss CB multicasts a LEAVESYNC message to all coordinators (including itself) and waits for a LEAVEOK from each of them[6]. Then it issues a NORMAL message whose memb field describes the leaving of mid.

Upon receiving a LEAVESYNC identifying a leaving group member mid, a coordinator C_k frees the entry for mid from all GTable items. Then it removes mid from the pending field of all elements of norm-buffer and removes the elements of the latter in which the pending field has become empty. Finally it responds to CB with a LEAVEOK.

Coordinators discard NEW messages carrying a mid not included in members. Such messages are duplicates of old messages issued by MHs that left the group (and perhaps re-joined with a different join-id). The boss CB discards JOIN messages carrying a mid included either in members or in member-cache. The former case is a duplicate sent during the last join attempt of a current group member, whereas the latter is a duplicate sent by a MH that has already joined and left the group (perhaps several times).

4.6.2 Actions at MSSs

Upon receiving a NORMAL message, a MSS stores this message in norm-cache and broadcasts it in the wireless cell. If the memb field is not null, MSS takes the following actions before broadcasting:

---

[5] The leaving field may be omitted.

[6] The protocol in [B98] manages dynamic membership with a stronger semantics, i.e., a leaving group member is required to deliver all NORMAL messages that the coordinator originated after receiving the JOIN and before receiving the LEAVE.

- If memb describes the joining of mid, then MSS allocates an entry for mid in all GTable items and sets each element of first[C_k][mid] equal to the corresponding sequence number in memb. Furthermore, if mid belongs to local, it inserts mid in local-m.
- Otherwise, memb describes the leaving of mid, MSS frees all GTable items allocated to mid, removes mid from local-m (if present) and aborts the sequence of TRANSFER to mid possibly in progress.

Upon receiving a NACK message, MSS checks whether an entry for the sending group member mid exists in first. If not, MSS discards the NACK otherwise it starts relaying the missing messages to mid, as follows. Let ($slow_k$, $shigh_k$) be the sequence numbers associated with C_k in the received NACK. MSS transfers messages originated by C_k with sequence numbers in [$tr\text{-}low_k$, $tr\text{-}high_k$], where $tr\text{-}low_k$ = max($slow_k$,first[Ck,mid]) and $tr\text{-}high_k$ = max($shigh_k$, cseq[C_k]) (it might be $shigh_k$ = 0).

A NACK might be received when MSS is already sending TRANSFER messages to mid. In this case, MSS handles the NACK as follows. Let [$slow_k$+1, $shigh_k$-1] be the sequence numbers of the messages originated by C_k that characterize the in-progress sequence. Let $s1_k$, $s2_k$ denote the pair field of C_k in the just received NACK message m: (I) if $s1_k \leq tr\text{-}low_k$, then MSS continues the in-progress sequence; otherwise, (II) MSS aborts the sequence and starts a new one according to the above rules.

4.6.3 Actions at group members

Assume for the moment that the MH is already a group member, i.e., it delivered the NORMAL message mj describing its joining and used the content of mj for initializing mystable and delivered. At the end of this section we shall describe how to ensure that MH indeed delivers mj.

A received NORMAL message m is handled by procedure HandleNormal() in Figure 4[7]. Boolean function Discardable(m) (line 2) returns true iff m has already been delivered by the executing group member (lines 21-22). 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.

Let predicate DLVD(C, s, mid) be true iff group member mid has delivered the message with sequence number s sent by coordinator C. Let predicate DLVDAll(C, s, mid) be true iff mid has delivered *all* messages sent by C with sequence number smaller than, or equal to, s. We indicate the executing group member by my-mid and use the dot notation for indicating fields of a message, e.g., m.order for the order field of message m. The delivery conditions are listed in Figure 3 along with the criterion for their selection. Translating delivery conditions in terms of mystable and myahead is obvious and omitted.

Selection	Delivery condition
m.order = F-Cast()	DLVD(m.Coord, m.cseq-mid, my-mid)
m.order = C-Cast()	DLVD(m.Coord, m.cseq-mid, my-mid) and ∀Ci, DLVDAll(Ci, m.Deliver[Ci], my-mid)
m.order = T-Cast()	DLVD(m.Coord, m.cseq-mid, my-mid) and DLVD(CB, m.cseqB - 1, my-mid) and ∀Ci, DLVDAll(Ci, m.Deliver[Ci], my-mid)
m.memb.flag = joining	DLVD(CB, m.cseqB - 1, my-mid)

Figure 3 Delivery conditions

A message that is not a duplicate but does not satisfy its delivery condition may be buffered (lines 8 and 32-34). The delivery of a message may make it possible that the delivery conditions of some buffered messages evaluate to true (lines 5, 12-20, 6, and 23-31). Function ScanBuffer() 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 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 MSS a NACK message, whose pair fields are selected as follows. 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

[7] This pseudo-code privileges clarity over efficiency.

is no such message:

$s1_k$ = mystable[Ck].

$s2_k$ = 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).

A MH wishing to become a group member starts listening to messages directed to the group and waits for receiving a NORMAL message from the boss CB. Upon receiving such message m, MH assigns mystable[CB] equal to the cseqB field in m and then sends the JOIN. At this point, it starts executing a slightly modified version of the previous protocol: (i) skip line 4 and 27 (i.e., it discards messages that should be delivered); (ii) ignore messages sent by coordinators other than CB; (iii) use delivery condition in the bottom row in Figure 3. In short, the joining MH listens to all messages issued by CB until receiving the NORMAL message mj describing its joining. At this point, MH initializes mystable and delivered with the sequence numbers in the memb field of mj, removes from myahead and mybuf all messages with sequence number smaller than those in mystable and switches to executing the ordinary protocol.

```
1  Procedure HandleNormal(m:MsgT); {
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 Procedure UpdateData (m:message); {
13   if m.cseq = mystable[m.Coord] + 1 then {
14     mystable[m.Coord] = mystable[m.Coord] + 1;
15     while mystable[m.Coord]+1 ∈ myahead[m.Coord] do {
16       remove mystable[m.Coord]+1 from
                                    myahead[m.Coord];
17       mystable[m.Coord] = mystable[m.Coord] + 1;
18     }
19   } else insert mystable[m.Coord] in myahead[m.Coord];
20 }
```

```
21 Function Discardable(m:MsgT): boolean;
22 { return(m.cseq ≤ mystable[m.Coord] or
          m.cseq ∈ myahead[m.Coord]); }

23 Procedure CheckBuffer();{
24   m1=ScanBuffer();              // Not shown
25   while m1<>null do {
26     extract m1 from mybuf;
27     deliver m1;
28     UpdateData(m1);
29     m1=ScanBuffer();
30   }
31 }

32 Procedure Buffer(m:MsgT);
33 { if (there is space for m in mybuf and
              m is not in mybuf) then
34    insert m in mybuf }
```

Figure 4 Main actions at group members

4.7 Discussion of delivery conditions

Condition DLVD(m.Coord, m.cseq-mid) obviously enforces FIFO Order among messages originated by m.mid. As an example, consider Figure 5 (left). Messages going up are tagged NEW, those going down are tagged NORMAL and some NORMAL messages are omitted for clarity. Intermediate message-exchange at MSSs 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.

Examples about the delivery condition of C-Cast(), are given in Figure 6. 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, DLVDAll(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 DLVD(m.Coord, m.cseq-mid), because all multicasts issued by mid are sequenced 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 5 (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.

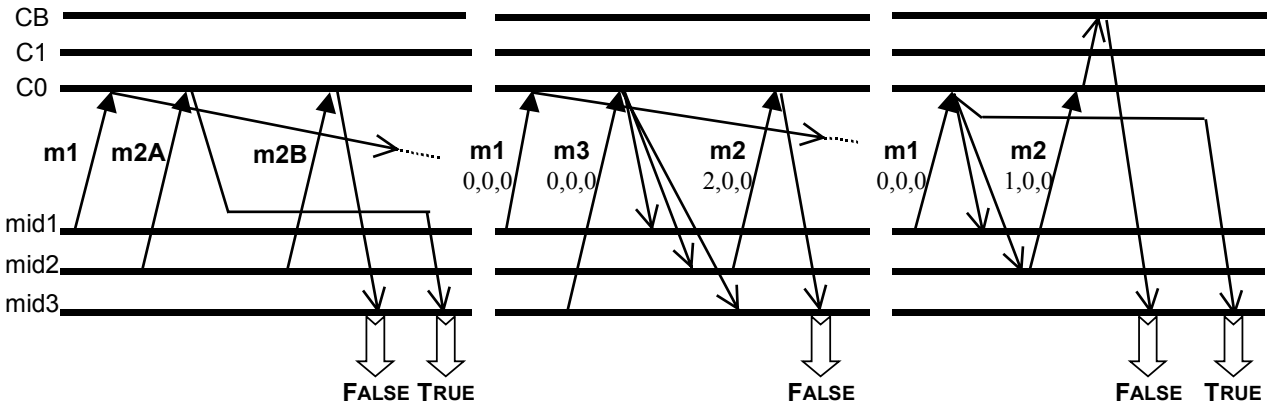

Figure 5 Examples for clarifying delivery conditions (see the text).

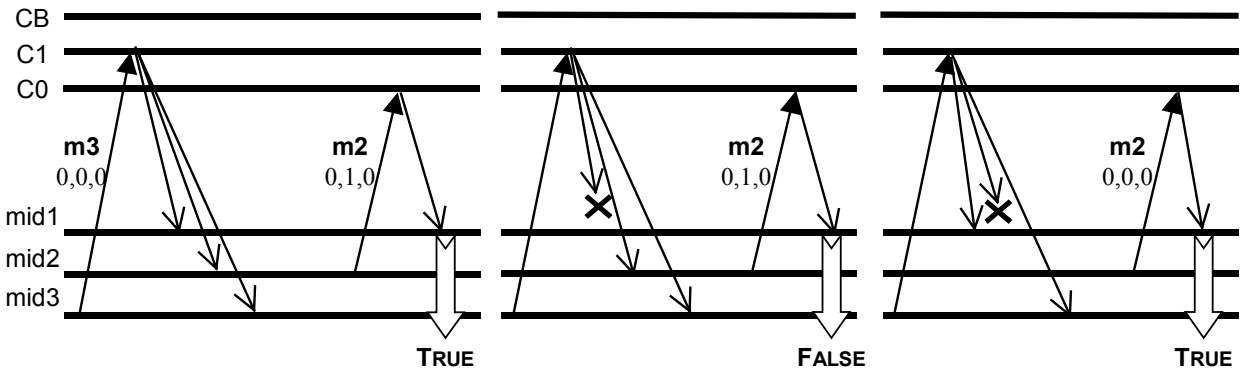

Figure 6 Examples of delivery conditions at mid1 upon receiving C-Cast(m2) issued by mid2.

The ultimate reason for this feature (called *inhibition* in **[AV97]**) 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 5 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 **[AV97, ABS99a]**, 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.

As for T-Cast(), the sub-condition DLVD(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 5 (right). 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.

Finally, the delivery condition for membership changes notifying a joining enforces in-order delivery of messages sent by CB. That is, it the same as T-Cast() but without the additional sub-conditions for enforcing Causal Order. It is so because the joining process has not sent any message yet and because the JOIN message arrives directly at CB without passing at any intermediate coordinator.

4.8 Involving only MSSs of non-empty cells

The subscribed set at coordinators will contain the identifiers of all MSSs covering the area of interest. The proposed protocol is amenable to a simple extension in which the composition of this set can change dynamically: (I) a MSSs that is not currently "subscribed" may ask coordinators to include it in the subscribed set whenever its cell becomes non-empty; (II) a subscribed MSS whose cell has remained empty for a while may "unsubscribe". We do not give all details for sake of brevity and refer the reader to **[B98]**, that considers the case of a single coordinator. Essentially, subscription and unsubscription consist of a remote procedure call from the MSSs to the boss CB. The processing of this call consists in updating the subscribed set at all coordinators, which can be done with one multicast on the wired network, from CB to coordinators.

When a group member enters a cell of a MSS that is not subscribed, it may perceive an increased latency of message delivery. However, this delay is of the order of time necessary for processing the related remote procedure call, that is small compared to the speed of users' movements. Correctness is not affected because, intuitively, the cell of the MSS being subscribed is equivalent to an uncovered area.

Observe that any reliable multicast protocol that does not *always* send *all* multicasts to *all* MSSs must exhibit a sort of start-up period when a group member enters an empty cell. Furthermore, other protocols offer similar functionality only by means of schemes that are more complex and more costly than that outlined here. For example, the extended form of the protocol in **[AB96]** uses a data structure, called *location view*, that is the set of MSSs whose cells contain group members. This location view is replicated at *all* MSSs in the view. Updates to the location view must be performed at *each* replica and must be serialized. Hand-off must be properly synchronized with location view management.

5 Simulation Environment

5.1 Overview

To analyze the performance of the protocol we developed a discrete event simulation model and implemented it in C++ language. The simulator supports a single multicast group with static composition. We did not implement dynamic membership as it would have increased the execution times of the simulation experiments without providing significant insights. The model includes three kinds of entities: group members, MSSs and coordinators (Figure 7). Messages received from the network are stored in the *Processing Queue* and then processed sequentially by the *Protocol Module*, that implements the proposed protocol. Actions in this module may involve transmitting a message along either the wired or the wireless network. In either case, when the pertinent network interface is busy, the message is temporarily stored in a *Transmission Queue*. The *Application Process* at group members merely generate messages. According to a common practice, time intervals between consecutive messages are random variables exponentially distributed[8] (i.e., the message generation process is Poisson).

[8] The exponential distribution function is $F(t)=1-e^{-rt}$ where r is the message rate, i.e., the number of messages generated per time unit.

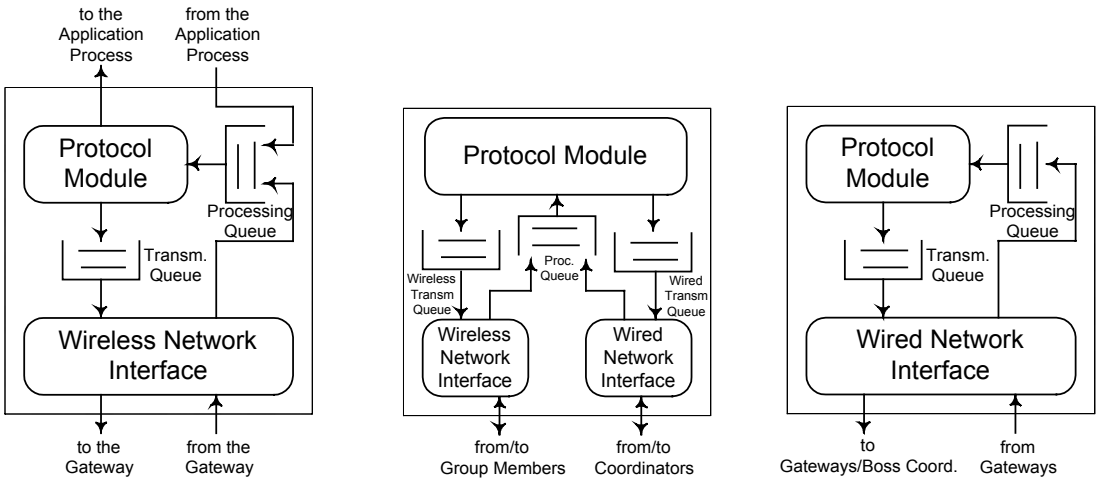

Figure 7: Simulation models of group member (left), MSS (middle) and coordinator (right).

Mobility is taken into account as follows. A group member remains in a cell for a random time interval exponentially distributed (as in **[AV97]**). Then it either switches to another cell or enters the uncovered area, based on a predefined probability. The new cell is selected independently of the current position, i.e., all cells have the same probability. A group member not in coverage remains so for a random time interval exponentially distributed.

To better understand the simulator and the result, it is important to consider the components of the end-to-end delay, i.e., the time experienced by a message along its way from a sending group member mid_s to a receiving group member mid_r. These components are shown in Figure 7 which refers to the case when no message loss occurs (relative lengths of time intervals does not reflect their actual values). When a message is lost, the end-to-end delay of that message includes a further component, the retransmission delay, which is the time necessary to detect and recover the lost message. Processing and buffering occur in the Protocol Module, the latter being the case when the message has to be delayed because it arrived out-of-order (e.g., in the `Mybuf` buffer at group members). Queuing for processing and for transmission occur in the Processing Queue and Transmission Queue, respectively. Transmission delays are the times necessary to actually put bits on the wireless/wired medium while propagation delay is the time it takes to these bits for reaching the intended destination.

In our simulation model we assumed that processing times are constant, but different at coordinators, MSSs and group members. Furthermore, these times depend on the message tag. The corresponding values, given later, were estimated by measuring the time spent by the simulator for executing the corresponding protocol actions[9], that are very close to those necessary in a full implementation.

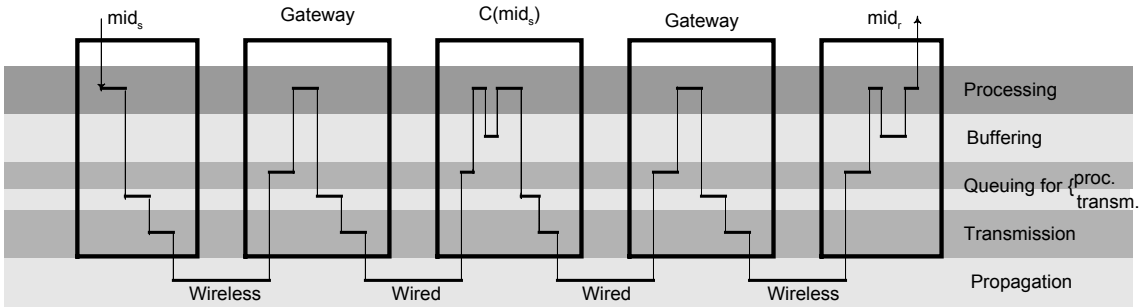

Figure 8: Delays experienced by a message from the sending group member (mid_s) to a receiving group member (mid_r).

Transmission and propagation times have been modeled as follows:

- Wired network: the transmission time is constant and follows from the value selected for the wired bandwidth, given later, whereas the propagation delay is a random variable with a uniform distribution in a specified range. Wired channels are assumed to preserve FIFO ordering.

[9] The simulator was running on a PC based on a 200 MHz Pentium Pro processor with 64 Mbytes of memory and Linux operating system and it was the only active user process.

- Wireless network: the transmission time is constant. Possible collisions at the MAC layer were taken into account by considering an available wireless bandwidth (1 Mbps) less than the nominal bandwidth of current WLANs (2 Mbps, see Section 5.3). The propagation time was assumed to be negligible due to the relatively small cell diameter assumed (see Section 5.3).

As an aside, it was assumed that the wireless network is inherently broadcast, i.e., a single wireless transmission suffices to send a given message to all group members in a cell.

Buffering and queuing times cannot be predicted in advance as they depend on a number of factors, e.g., message generation rate, number of sending group members, etc. One of the valuable results of our simulations are estimates of buffering times and queuing times. We anticipate that *queuing delays at MSSs*, and specifically at the wireless interface between MSSs and group members, are predominant. This result is particularly significant because this delay component is due to the mismatch between the wired bandwidth and the wireless bandwidth, hence MSSs are likely to be the ultimate bottleneck of any multicast protocol for distributed mobile systems.

5.2 Performance Measures

The simulator has been instrumented so as to evaluate the following performance indexes:

- *average end-to-end delay* (or *average latency*): average time elapsed from the instant at which a multicast message is generated at the sending group member mid_s to the instant at which the same message is delivered by a destination group member mid_r.
- *percentage of retransmitted messages*: number of messages retransmitted by MSSs (i.e., solicited by NACK) over the number of messages delivered by group members.
- *percentage of duplicates*: number of messages discarded as duplicates over the number of messages delivered by group members.
- *average realignment delay*: average time interval between the time instant at which the group member reconnects to the time instant at which all the messages missed during the disconnection period have been delivered. This index is meaningful only in experiments where the coverage is not complete.

Latency is the main performance index of our interest. Percentage of retransmissions and duplicates allow us to gain insight into the basic design choice of not requiring hand-off in the wired network upon movements of group members — retransmissions and duplicates are the potential drawbacks of absence of hand-off[10]. Realignment delay quantifies the performance loss perceived by users that frequently disconnect and reconnect.

5.3 Parameter Setting

The behavior of the protocol is, in principle, influenced by a very large set of parameters. For this reason, the simulator has been designed to make it very flexible and it is almost fully parameterized. The set of parameters can be logically partitioned in three subsets, as follows:

- *System parameters*: those that depend on the operating environment, e.g., number of MSSs, properties of communication channels and alike (Table 1).
- *Group member parameters*: those that depend on the behavior of group members, e.g., message generation rate, mobility pattern (Table 1).
- *Protocol parameters*: those that depend on the protocol actions, e.g., message size, processing times, size of `Mybuf` and alike (Table2). These parameters can be subdivided further, by distinguishing those that are fixed (i.e., determined by the underlying computing platform) from those that can be tuned depending on the desired trade-offs.

Of course, even if one considered only few values for the main parameters, the number of possible scenarios would be unmanageable. For this reason, we define a *Basic Scenario* with values that identify an utilization environment expected to be "realistic". Then, we analyze the influence of the main parameters by varying the value of one parameter at a time with respect to the Basic Scenario. Values in Table 1 and Table2 are those characterizing the Basic Scenario. We briefly discuss the main choices about these values in the

[10] Actually, retransmissions and duplicates are due not only to mobility but also to, respectively, unreliability of the wireless channel and NACK mechanism (see Section 5.3). Other multicast protocols assume that the wireless channel is reliable.

following, more details can be found in **[ABS99a]**.

Parameter		Value	
Number of MSSs	(N_g)	40	
Wired bandwidth		10	Mbps
Wireless bandwidth		1	Mbps
Wired propagation delay		[0.5 - 2.5]	msec
Wireless propagation delay		0.0	msec
Message loss probability	(P_{loss})	0.001	
ACK message loss prob.		0.0	

Parameter		Value	
Number of group members	(N_r)	100	
Number of senders	(N_s)	10	
Message rate		8	mess/sec
Cell permanency time	(T_{cell})	5	sec
Out-of-coverage perm. time	(T_{out})	0	sec
Out-of-coverage probability	(P_{out})	0	

Table 1: System parameters (left) and group member parameters (right)

We considered an operating environment similar to a small campus or building. We assumed a wired bandwidth of 10 Mbps with propagation delay uniformly distributed within the range [0.5 - 2.5] msec. These values correspond to those obtained in the internetwork of the campus of the Faculty of Engineering of the University of Pisa for messages of size similar to those involved in the simulator. We assumed a wireless bandwidth of 1 Mbps which is less than the nominal bandwidth of current Wireless LANs (2 Mbps) to take into account the bandwidth wastage due to collisions at the MAC layer[11]. Propagation delays are negligible as wireless cells are supposed to be very small, approximately ten meters. Simulation results for wired and wireless bandwidth of 100 Mbps and 10 Mbps, respectively, can be found in **[ABS99a]**.

Group members generate messages according to a Poisson process (i.e., time intervals between consecutive messages are random variables exponentially distributed) with average rate of 8 messages/sec (32 Kbps). The resulting aggregate message rate is 80 messages/sec (320 Kbps) because there are 10 sending group members. The *cell permanency time* and the *out-of-coverage permanency time* are, respectively, the average time interval that a group member remains in a wireless cell or in the uncovered area (recall that these times are exponentially distributed). The *out-of-coverage probability* is the probability that a group member enters the uncovered area. The cell permanency time of 5 sec corresponds to a person walking (7 Km/h) across cells of approximately 10 m. The out-of-coverage probability is set to 0 in the Basic Scenario, i.e., complete coverage, but we performed experiments where this probability is not null and the out-of-coverage permanency time progressively increases.

For sake of simplicity, we assumed that all messages are equal-sized, irrespective of the tag. The only exception is for ACK messages, that were assumed to be smaller. Sizes in Table2 include control information and actual payload, if any. Processing times were selected as described in **[ABS99a]**, i.e., by measuring the corresponding CPU time in the simulator.

Parameter	Value	
NEW/NORMAL/TRANSFER message size	512	bytes
ACK message size	50	bytes
NACK message size	512	bytes
FETCH message size	512	bytes
Beacon message size	64	bytes
Processing time at group members	0	µsec
Processing time of NEW msg at coord.	30	µsec
Processing time of FETCH msg at coord.	9000	µsec
Processing time of NORMAL msg at the Boss coord.	25	µsec
Processing time of NACK msg at MSSs	1000	µsec
Processing time of NEW msg at MSSs	25	µsec
Processing time of NORMAL msg at MSSs	290	µsec

Parameter		Value	
Number of coordinators	(N_c)	2	
Service Ratio	(R_{serv})	10	
`Mybuf` buffer size		1000	msgs
MSS cache size		1000	Msgs
Beacon period		100	msec
Retransmission timeout		40	msec
NACK timeout		300	msec

Table2: Protocol parameters: fixed (left) and tunable (right)

6 Results

This section is devoted to the discussion of the simulation results[12]. These results have been obtained by assuming that all messages are issued by the `T-Cast()` primitive (i.e., they require Total order). We also performed experiments with all messages requiring Fifo or Causal order. The results obtained were similar

[11] The probability of collisions, and hence, bandwidth wastage, depends on the operating conditions. In [AL99] it is shown that the achievable throughput for a IEEE 802.11 WLANs is in the order of half of the nominal bandwidth.

[12] Results have been estimated by using the independent replication method and assuming a confidence level of 90% [LK82].

to those reported below. Additional results can be found in **[ABS99a]** where, in particular, it is shown that unnecessary buffering delays due to inhibition (see Section 4.7) are negligible.

6.1 Scalability

In this section we assess the scalability of the protocol, i.e., how the protocol performs when the number of sending or receiving group members becomes larger and larger. First we vary the number of receiving group members (N_r) while maintaining the number of sending group members (N_s) constant. Then, we fix the number of receivers and increase the number of senders.

6.1.1 Scalability with the number of receivers

Figure 10 shows that the average end-to-end delay is not significantly affected by the number of receivers (N_r) and is slightly greater than the minimum latency. The latter denotes the minimum time needed by messages to reach the destination group member and only includes transmission, propagation and processing delays. In practice, the minimum latency is experienced when (**i**) the workload both in the wired and in the wireless networks is very low (so that queuing delays can be neglected); (**ii**) no message gets lost (i.e., the retransmission delay is null); (**iii**) at the destination group member the delivery conditions is always satisfied (i.e., the buffering delay is null).

The results in Figure 9 shows that the protocol scales very well with the number of receivers: the average end-to-end delay remains almost constant not only when the number of receivers is less than or equal to the number of MSSs (N_g=40) but even when it is largely greater. This is a valuable property for a multicast protocol. Several features of the proposed protocol contribute to make it so well scalable in terms of receivers, including: a single wireless transmission for broadcasting within each cell (unlike **[AB96]**); absence of hand-off upon cell switching; cache of past messages at MSSs. The protocol in **[ARR97]** definitely does not exhibit this feature, as its latency quickly increases even with a few receivers. We are not aware of similar measurements for the protocol in **[AB96]**[13].

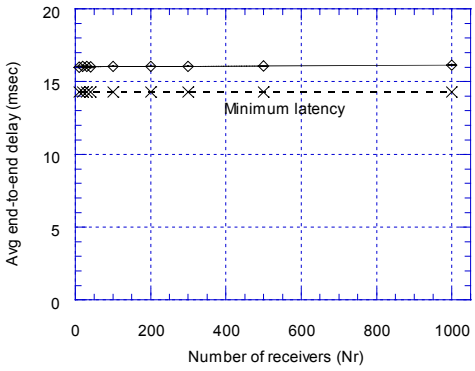

Figure 10: Scalability in terms of receiving group members

6.1.2 Scalability with the number of senders

Another important property to evaluate is the scalability with respect to the number of senders (N_s). When the value of N_s increases the aggregate message rate, i.e., the workload in the network increases accordingly. From Figure 11-left it appears that the average end-to-end delay remains close to the minimum latency while the workload is less than 80% of the wireless capacity but it dramatically goes up as the workload tends to saturate the wireless network. This behavior can be easily justified by looking at the components of the average end-to-end delay shown in Figure 11-right.

Apart from the minimum latency (transmission + propagation + processing times) all the components increase as the number of senders goes up. In fact, when the aggregate message rate increases messages experience larger queuing times in the Processing and Transmission Queues at coordinators and MSSs. Furthermore, the number of messages to buffer because of message losses increases. Finally, NACK and TRANSFER messages experience larger delays and, hence, the time to recover lost messages (i.e., the retransmission delay) increases.

[13] We have preliminary simulation results for this protocol showing that the average latency greatly increases with the number of receivers.

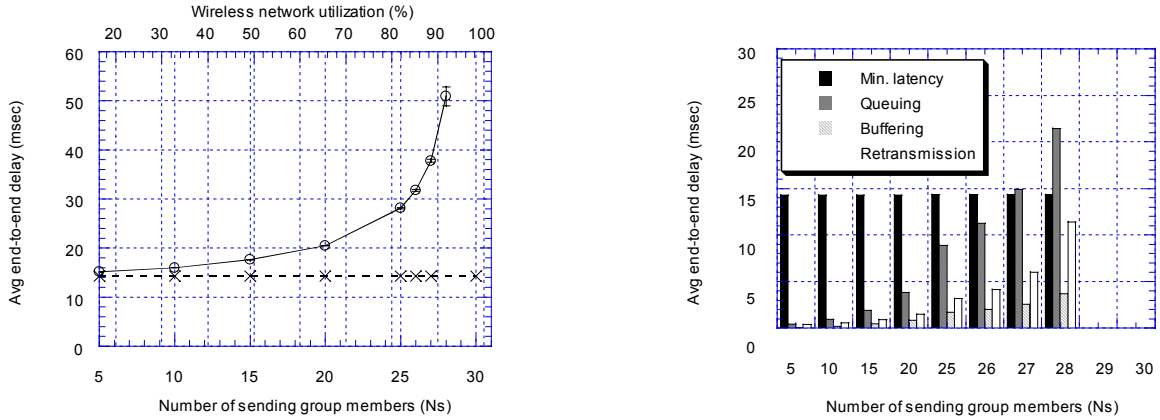

Figure 11: Average end to end delay vs. number of sending group members N_s (left). Delay components (right).

6.1.3 Potential Bottlenecks

An important issue related to the scalability with the number of senders is the problem of possible bottlenecks. A potential critical point might be the centralized Team Boss coordinator which processes all the messages issued by the `T-Cast()` primitive and, hence, might become overloaded when the aggregate message rate become larger and larger. However, our simulation results show that the likely bottleneck of the system, rather than the Team Boss, appears to be the wireless interface between MSSs and group members. Table3 reports (in addition to the average end-to-end delay) the average total and queuing delays experienced at different points of the system (destination MSS, coordinator and Team Boss) for aggregate message (bit) rates up to 224 messages/sec (918 Kbps[14]).

It can be seen that: (I) the average total delays at the coordinator and at the boss are practically negligible in comparison to that at MSS; (II) the total delay at MSS (and specifically the queuing component) tends to become the predominant factor of the overall latency. Simulation results also show that this delay is mostly experienced in the transmission queue. This result is clearly due to the mismatch between the bandwidth of the wired and wireless networks.

N_s	Aggregate message rate (mess/sec)	Aggr. bit rate (Kbps)	Average end-to-end delay (msec)	Destination MSS		Coordinator		Team Boss	
				Average total delay (msec)	Average queuing delay (msec)	Average total delay (msec)	Average queuing delay (msec)	Average total delay (msec)	Average queuing delay (msec)
15	120	480	17.61	6.27	1.88	0.44	0.00	0.45	0.01
20	160	640	20.49	8.10	3.80	0.44	0.00	0.45	0.01
25	200	800	28.11	12.99	8.60	0.44	0.00	0.46	0.02
26	208	852	31.77	14.17	10.78	0.44	0.00	0.46	0.02
27	216	884	37.81	18.46	14.07	0.44	0.00	0.46	0.02
28	224	918	50.91	23.95	19.56	0.44	0.00	0.46	0.02

Table3: Average delays experienced at different points of the system.

We argue that even the limitation due to the wireless interface between MSSs 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) **[AB96]**, is likely to exhibit a bottleneck at MSSs. The observation that MSSs are likely to constitute an intrinsic potential bottleneck is particularly useful, because it allows using a simple, logically centralized coordinator-based protocol.

6.2 Effects of host mobility

Host mobility can be quantified either by the average cell permanency time (T_{cell}) or by its inverse, i.e., the average number of cell switches per time unit and per group member. Table 4 reports the values of T_{cell} used in the simulation experiments and the corresponding frequencies of cell switching. Some of these values may

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

be not realistic (e.g., T_{cell}<1) but are considered in order to stress the protocol in very critical situations. Table 4 also provides for each value of T_{cell} the corresponding speed of a group member moving at a uniform speed in cells of approximately 10 m.

T_{cell} (sec)	Cell switches per sec ($1/T_{cell}$) (sec^{-1})	Speed (Km/sec)
50	0.02	0.72
5	0.2	7.2
2	0.5	18
1	1.0	36
0.5	2.0	72

Table 4: Relationship between average cell permanency time, frequency of cell switching and group member's speed.

Figure 12(left) shows that, as expected, the average end-to-end delay tends to increase as the host mobility grows. This behavior can be justified by observing that when the frequency of cell switching increases the number of missed messages increases accordingly. This causes a greater retransmission delay since missed messages need to be retransmitted and a greater buffering delay since a larger number of messages needs to be buffered at the group member. However, it clearly appears that the influence of host mobility on the average end-to-end delay is very limited. This is a desirable feature for a protocol for mobile systems. Observe that if a host occasionally moves very quickly it experiences a performance penalty but the protocol remains correct. When the host's speed comes back to normal values the performance penalty disappears.

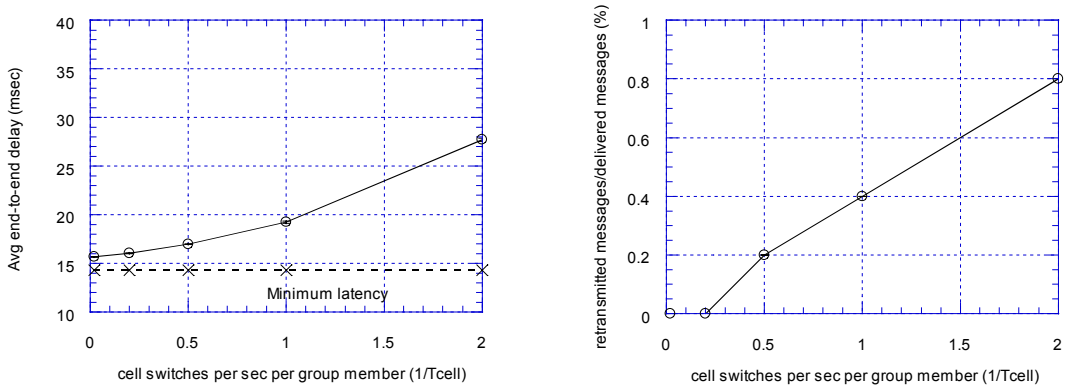

Figure 12: Average end to end delay vs. host mobility (left). Percentage of duplicate messages vs. host mobility (right).

We also estimated the percentages of retransmitted and duplicate messages (over the number of delivered messages) caused by host mobility[15]. Since retransmitted messages consume wireless bandwidth, processing power and battery energy their fraction gives an indication of the additional resource consumption introduced by not using the hand-off procedure. Duplicate messages do not consumes wireless bandwidth[16] but their reception causes a slightly increase in the consumption of processing power and battery energy at the group member since they have to be processed (and discarded) by the protocol entity at the group member. Hence, their percentage gives and indication of the additional resource consumption at the group member.

The percentage of retransmitted messages as a function of host mobility is shown in Figure 12-right (the percentage of duplicate messages has a similar behavior and is not reported here). Even when host mobility is high the percentage of retransmitted (duplicate) messages remains less than 1%, i.e., very limited. The above results show that it is probably a good idea to not build multicast protocols over the hand-off procedure. Hand-off would prevent the arrival of duplicate messages at group members and useless transmissions of messages caused by host mobility, but the above results show that insisting on that is probably not worthwhile.

[15] Message retransmissions may also be caused by the unreliability of the wireless channel. Similarly, duplicate messages can also arise due to the NACK mechanism. However, throughout this section we shall refer to retransmitted and duplicate messages caused by host mobility only.

[16] Remember that the wireless medium is intrinsically broadcast and, hence, messages are received by all the mobiles in the cell and, if not destined to that mobile, discarded at some layer in the protocol stack.

6.3 Influence of the Buffer Size

In Section 5.3 we set the `Mybuf` buffer size to 1024 512-bytes messages (512 Kbytes). This very large size is necessary because we intend to stress the protocol in very critical conditions, in particular, in presence of very large message loss rates (see Section 6.4) and in wireless networks with incomplete coverage where group members can experience large disconnection periods (see Section 6.5). However, Figure 13-left - which refers to the Basic Scenario – shows that in normal conditions a buffer capacity of 16-32 messages (8-16 Kbytes) can be sufficient. Moreover, it may be worthwhile to point out once again that the `Mybuf` buffer only improves performance but the protocol would work correctly even without any buffer.

6.4 Effects of network unreliability

In addition to host mobility, unreliability and incomplete coverage of the wireless networks are additional sources of message losses. The effects of host mobility have been considered in Section 6.2. In this section we analyze the protocol sensitivity to the unreliability of the wireless links while the next sessions will be devoted to analyze the influence of incomplete coverage.

Figure 13-right shows that, as expected, the average end-to-end delay increases as the message loss probability of wireless cells (P_{loss}) grows. In fact, an increase in the value of P_{loss} implies a larger number of messages to retransmit and a larger number of messages to buffer at the group member (in the `Mybuf` buffer) waiting for the recovery of previous lost messages. Furthermore, retransmitted messages increase the workload in the wireless network and causes greater queuing delays.

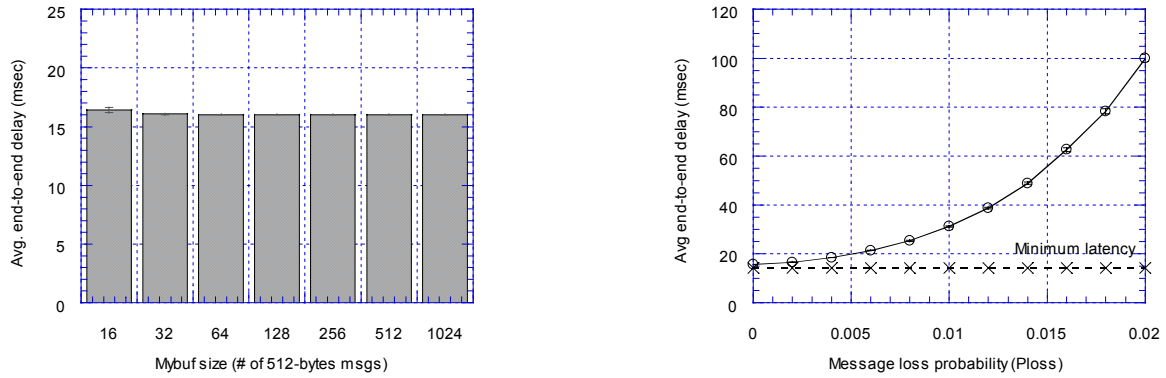

Figure 13: Average end-to-end delay as a function of the Mybuf buffer in the Basic Scenario (left). Average end-to-end delay as a function of message loss probability (right).

Message loss probabilities considered in Figure 13-right are limited to the range [0, 0.02] which is typical for an in-building environment without sources of interference **[XP99, ES96]**. However, we performed additional experiments with values of P_{loss} up to 0.1 (1 message out of ten is lost) and still observed the trend shown in Figure 13-right. The average end-to-end delay can become extremely large when the wireless communications are very unreliable. Indeed, this is not a peculiarity of our protocol: if messages are lost very frequently and have to be recovered the latency necessarily increases.

6.5 Effects of incomplete coverage

To analyze the influence of incomplete coverage we divided group members into two classes: members belonging to the first class never disconnects (i.e., their out-of-coverage probability, P_{out}, is equal to zero) while those belonging to the second class are expected to experience temporary disconnection periods. For simplicity, hereafter, group members belonging to the above classes will be referred to as *non-disconnecting* and *disconnecting* group members, respectively.

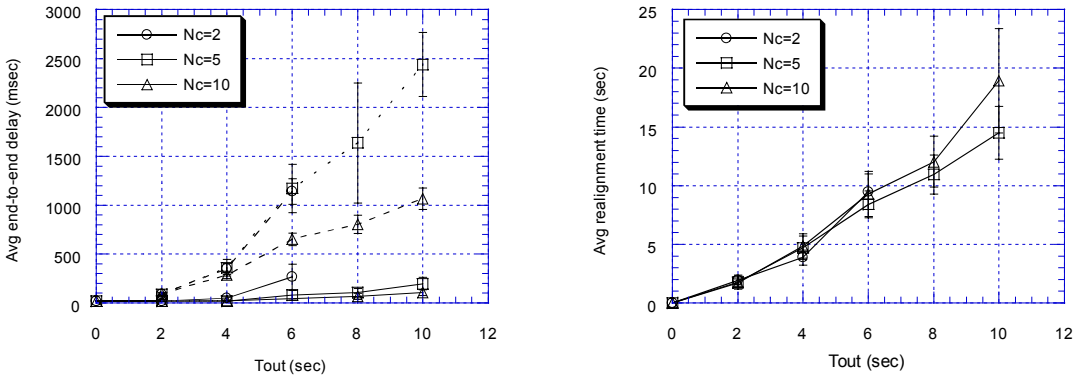

Figure 14: Average end to end delay (left) and average re-alignment time (right) vs. average out-of-coverage permanency time (T_{out}) for disconnecting (dashed lines) and non-disconnecting (continuous lines) group members.

The incomplete coverage can be quantified by the parameter T_{out}, i.e., the average time a group member remains disconnected. Figure 14-left shows the average end-to-end delay as a function of the value of T_{out} for disconnecting (dashed lines) and non-disconnecting (continuous lines) group members, respectively. As expected, the average end-to-end delay of disconnecting members increases as the average disconnection time goes up. However, also non-disconnecting members are influenced by the value of T_{out}. This can be justified as follows. Upon reentering the covered area a disconnecting member realizes that a relatively large number of messages have been missed and requires them to the current MSS. It may happen that some of the requested messages are no longer in the cache of the MSS and have to be fetched from the coordinator associated with the message originator. Hence, a greater T_{out} value implies greater queuing delays at coordinators and (destination) MSSs which affect both disconnecting and non-disconnecting members. The increase in the end-to-end delay is more evident for disconnecting members because (i) some messages need to be fetched from the coordinators; and (ii) MSSs manages disconnection related messages with lower priority with respect to other messages.

Figure 14 also shows that increasing the number of coordinators (N_c) is beneficial for both classes of members since it reduces the number of messages each coordinator has to process. In particular, it is possible to find a number of coordinators such that the average end-to-end delay of non-disconnecting group members grows very slightly in spite of the increase in the value of T_{out}. A similar effect can be obtained by increasing the size of MSS caches. In the latter case what is reduced is the probability that a message missed during a disconnection period is not in the MSS cache. Finally, the average end-to-end delay of non-disconnecting members could be reduced - but at damage of disconnecting members - by selecting a greater value for the service ratio R_{serv}, thus reducing the priority of disconnection related messages with respect to other messages.

Figure 14-right shows the average realignment time (i.e., the average time a group member needs to recover messages missed during a disconnection period) as a function of the value of T_{out}. This time is comparable with the disconnection duration and it is not significantly influenced by the number of coordinators. In fact, queuing delays at coordinators gives a small contribution to the realignment time especially in the case of long disconnection periods.

An interesting question is whether incomplete coverage can move the potential bottleneck from MSSs to coordinators. The rationale behind the question is that coordinators manage FETCH requests coming from MSSs and, hence, they might collapse if the traffic originated by disconnection periods becomes predominant. However, it has been just observed that the workload on the coordinator can be alleviated by increasing either the number of coordinators or the cache size at MSSs. On the other hand, messages missed by group members during disconnection periods have to be retransmitted by MSSs. This means that incomplete coverage certainly increases the workload at MSSs. Therefore, MSSs are expected to be potential bottlenecks even with incomplete coverage.

7 Conclusions

We have presented and evaluated a reliable multicast protocol for distributed mobile systems that supports dynamic membership and accommodates neatly three increasingly strong delivery ordering guarantees:

FIFO, Causal and Total. The system model assumed is quite general and includes incomplete spatial coverage of the wireless network while no limitations is posed on the mobility pattern of group members. Movements of MHs do not imply the exchange of any message in the wired network and, in particular, cell switching does not require any interaction between MSSs. The protocol has been designed by keeping in mind that MHs may have scarce resources. For example, the size of data structures at MHs and of message headers do not depend on the number of MHs. Furthermore, MHs do not acknowledge every received multicast individually.

Simulation results have shown that the protocol scales well and that the main limiting factor is the waiting time spent for the wireless transmission at MSSs. Since this queuing delay is due to the mismatch between the bandwidth of the wired network and the wireless network, this factor appears to be a necessary consequence of the underlying computing platform rather than a peculiarity of our protocol. From another point of view, the fact that queuing delays at MSSs are the likely bottleneck of any similar protocol implies that our simple coordinator-based architecture is indeed sound and practical. Another important result is that absence of hand-off does not introduce significant costs in terms of duplicate messages or useless retransmissions. It follows that the ability to accommodate efficiently frequent movements of a large number of MHs comes at almost no cost.

References

[AB96] 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.

[ABS99a] G. Anastasi, A. Bartoli, F. Spadoni, "A flexible multicast protocol for distributed mobile systems: design and evaluation", MOSAICO Project Technical Report PI-DII/1/99, February 1999. (http://www.iet.unipi.it/~anastasi/papers/tr99-1.pdf.gz).

[ABS99b] G. Anastasi, A. Bartoli, F. Spadoni "Group Multicast in Distributed Mobile Systems with Unreliable Wireless Network", Proceedings of the *18th IEEE Symposium on Reliable Distributed Systems (SRDS'99),* Lausanne (CH), October 18-21, 1999 (currently available at http://www.iet.unipi.it/~anastasi/papers/srds99.pdf.gz).

[AL99] G. Anastasi, L. Lenzini, "QoS Provided by the IEEE 802.11 Wireless LAN to Advanced Data Applications: a Simulation Analysis", *ACM/Baltzer Wireless Networks*, to appear (currently available at http://www.iet.unipi.it/~anastasi/papers/winet.ps.gz).

[ARR97] V. Aravamudhan, K. Ratnam, S. Rangajaran, "An Efficient Multicast Protocol for PCS Networks", *ACM/Baltzer Mobile Networks and Applications*, vol.2, n.4, pp. 333-344, 1997.

[AV97] S. Alagar, S. Venkatesan, "Causal Ordering in Distributed Mobile Systems", *IEEE Transactions on Computers*, 46 (3), March 1997, pp. 353-361.

[B98] 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.

[BPT96] P. Bhagwat, C. Perkins, S. Tripathi, "Network Layer Mobility: An Architecture and Survey", *IEEE Personal Communications*, June 1996, pp. 54-64.

[BSS91] 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.

[CP98] R. Caceres, V. Padmanabhan, "Fast and Scalable Wireless Handoffs in Support of Mobile Internet Audio", *ACM/Baltzer Mobile Networks and Applications*, vol.3, n.4, pp. 351—363, 1998.

[CWBM98] V. Chikarmane, C. Williamson, R. Bunt, W. Mackrell, "Multicast Support for Mobile Hosts Using Mobile IP: Design Issues and Proposed Architecture", *ACM/Baltzer Mobile Networks and Applications*, vol.3, n.4, pp. 365-379, 1998.

[DC90] S. Deering, D. Cheriton, "Multicast routing in datagram internetworks and extended LANs", *ACM Transactions on Computer Systems*, vol.8, n.2, May 1990, pp.85-110.

[ES96] D. Eckhardt, P. Steenkiste, "Measurement and Analysis of the Error Characteristics of an In-Building Wireless Network", *Proc. of ACM SIGCOMM'96,* August 1996.

[FZ94] G.H. Forman, J. Zahorjan, "The challenges of mobile computing", *IEEE Computer*, vol.27, n.4,

pp.38-47, April 1994.

[GS94] R. Ghai, S. Singh, "An architecture and communication protocol for picocellular networks", *IEEE Personal Communications*, Third Quarter 1994, pp.36-46.

[HR94] H. Attiya, R. Rappoport, "The level of handshake required for establishing a connection", *Distributed Algorithms*, Lecture Notes in Computer Science 857, Springer Verlag 1994, pp.179-193.

[HT93] V. Hadzilacos, S. Toueg, "Fault-tolerant broadcasts and related problems", in *Distributed Systems (2nd edition)*, Sape Mullender ed., ACM Press 1993.

[KT96] F. Kaashoek, A. Tanenbaum, "An evaluation of the Amoeba group communication system", *Proc. of the 16-th IEEE International Conference of Distributed Computing Systems*, May 1996, pp.436-447.

[L78] 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.

[L93] B. Lampson, "Reliable messages and connection establishment", in *Distributed Systems (2nd edition)*, Sape Mullender ed., ACM Press 1993.

[LK82] A. M. Law, W. D. Kelton, "Simulation Modeling and Analysis", McGraw-Hill Book Company, 1982.

[MAO92] K.S. Meier-Hellstern, E. Alonso, D.R. O'Neil, "The Use of GSM to Support High Density Personal Communication", *Record of the IEEE International Conference on Telecommunications*, June 1992.

[MDC93] B. Marsh, F. Douglis, R. Caceres, "Systems Issues in Mobile Computing", Technical Report MITL-TR-50-93, Matsushita Information Technology Laboratory, 1993.

[P96] C. Perkins, "IP Mobility Support", RFC 2002, Mobile IP Working Group, October 1996.

[PRS96] 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.

[RST91] M. Raynal, A. Schiper, S. Toueg, "The causal ordering abstraction and a simple way to implement it", *Information Processing Letters* vol. 39, n.6, September, 1991, pp. 343--350.

[XP97] G. Xylomenos and G. Polyzos, "IP Multicast for Mobile Hosts", *IEEE Communications Review*, January 1997, pp. 54—58.

[XP99] G. Xylomenos, G. C. Polyzos, "TCP and UDP Performance over a Wireless LAN", *Proc. of IEEE INFOCOM'99, 1999.*

[YHH97] L. Yen, T. Huang, S. Hwang, "A Protocol for Causally Ordered Message Delivery in Mobile Computing Systems", *ACM/Baltzer Mobile Networks and Applications*, vol.2, n.4, pp. 365—372, 1997.