Comparing Two Reliable Multicast Protocols for Mobile Computing

Mateus de Freitas Ribeiro

Instituto de Matemática e Estatística -
Universidade de São Paulo
Rua do Matão 1010 - 05508-900 - São Paulo - Brazil
mateus@ime.usp.br

Markus Endler

Departamento de Informática - PUC Rio -
Marquês de São Vicente 225 - Gávea
22453-900 - Rio de Janeiro - Brazil
endler@inf.puc-rio.br

Abstract

As networks with mobile devices becorne commonplace, many new applications for those networks arisc, including some that require coordination among groups of mobile clients. One basic tool for implementing coordination is reliable multicast, where delivery of a multicast message is atomic, i.e. cither all or none of the group members deliver the message. While several multicast protocols have been proposed for mobile networks, only a few works have considered reliable multicats.

In this paper we present and compare two protocols based on Two-Phase-Commit that implement reliable multicast for structured mobile networks. Protocol iAM²C is a variant of protocol AM²C that employs a two-level hierarchical location management scheme to locate and route messages to the mobile hosts addressed by a multicast. Although hierarchical location management is not new in the context of mobile and cellular networks, we are unaware of any other work which combines hierarchical location management with protocols for reliable multicast.

We have prototyped, simulated and evaluated both protocols using the MobiCS simulation enviromment. Our experiments indicate that despite some overhead incurred by the location management and the additional level of message redirection, iAM²C is more efficient than the AM²C protocol and scales well with the size of the wired network infra-structure.

Keywords — *Reliable Multicast, Two-Phase-Commit, Mobile Computing, Protocol Simulation*

1 Introduction

As networks with mobile devices become commonplace, many new applications for those networks arise, including ones requiring coordination among groups of mobile clients. Examples of such services are strategic mission planning and execution, optimization in vehicle movement, cooperative work, and many others.

In order to support coordinated actions, a mobile host must be able to communicate reliably with a group of related mobile hosts. Some applications even require a consistent view of a distributed state, composed of the local states at the hosts of the mobile group. For such applications, delivery of a multicast message should be atomic, meaning that a message is either accepted by all mobile hosts of the group or by none of them.

In the following, we give two examples of applications with such strict consistency requirements.

Distributed Agreement A group of friends communicating through their mobile devices plan to meet for a movie and need to reach an agreement on the film, the theater and the time to meet, since tickets are to be bought in advance. Consider that the main restriction here is that *all* friends of the group must agree, i.e. no proper sub-group should meet, since otherwise those left out may become jealous. So each of the friends in turn makes a proposal, and eventually the group reaches an agreement. Reliable multicast is of great value for such distributed decision-making, since after each proposal (a multicast), each of the parties involved learns whether the group has reached an agreement or not. In fact, if for some reason one of the friends is not reachable during a period of time, then no proposal submitted during this period of time should be committed. Moreover, if he/she becomes disconnected right

*Partly supported by CNPq (M.Sc. Scholarship). Current Email: mateus.ribeiro@dedalus.com.br

after having voted, he/she should be informed about the outcome of the corresponding proposal.

Coordinated Position Tracking Members of a Coordinated Action Group (e.g. Police Task Force, Fire Brigade, Traffic Control Department) need a consistent and up-to-date view of the current locations of each member of the group. For example, assume that each member of the group carries a mobile device (capable of local position detection through GPS) through which he/she is able to query the current location of all other members. If the coordinated action requires up-to-date and accurate data about each member's current location, then each query must be replied by all members. If one of the members is not reachable then all other members must be aware that the result received for the query may not contain up-to-date information.

In order to support the implementation of such forms of coordination, we have investigated how to adapt Two-Phase-Commit (2PC) protocols to mobile networks. Our first protocol, called AM^2C [9], was an extension of Acharya and Badrinath's MCAST [2] reliable multicast protocol, but AM^2C was not scalable. In this article we present a variant of this protocol, called the *Indirect* AM^2C (iAM^2C) [8], which adopts a two-level approach for managing the mobile hosts' locations. Essentially, iAM^2C is a Two Phase Commit (with an additional clean-up phase) executed among the hosts in the static part of the network, and where Mobility Support Stations act as simple network access points for the mobile hosts.

Hierarchical location management schemes for tracking and routing in mobile networks are not novel [14], and in fact have been employed in past and current cellular networks, such as GSM, CDMA2000 or UMTS. However, we have not seen any other work which combines hierarchical location management with protocols for reliable multicast, i.e. which have the property of atomic message delivery (i.e. *all-or-nothing* delivery).

The main focus of this article is on showing how iAM^2C guarantees this property, rather than discuss possible orders of multicast message delivery. While total order can be added to 2PC protocols in a straight-forward manner, FIFO and causal delivery orders could be directly incorporated into our protocols in a similar way as proposed elsewhere [15].

We have implemented and simulated both iAM^2C and AM^2C protocols using the *Mobi*CS prototyping environment [7,17], and have analyzed and compared through simulation the message complexity of both protocols for different network configurations and mobility rates.

The remainder of this paper is organized as follows: In the next section we describe the system model and the main assumptions. In Section 3 we outline the AM^2C protocol, and explain iAM^2C in detail. Section 4 contains the complexity analysis of the protocols. In Section 5 we present the *Mobi*CS environment and results obtained from the protocol simulations. Section 6 mentions related work and finally, in section 7 we draw some conclusions and mention our future work.

2 System Model and Main Assumptions

We assume a structured mobile network, consisting of *Static* and *Mobile hosts (Mh)*. Among the static hosts, some are *Mobility Support Stations (Mss)*, which are the network access points for the *Mh*s, and others are *Intermediate Message Servers* (*IMS*s), which are repositories of multicast messages and which control the delivery of those messages to the *Mh*s located in several cells. The static hosts are assumed not to fail, and are linked with each other through a static, reliable computer network. Each *Mss* implicitly defines a geographic region, called *cell*, where it is able to communicate with a set (*local_Mhs*) of mobile hosts located in the cell. The set of all cells of a system is partitioned into disjoint groups, called *domains*, where each *domain* has a single *IMS* responsible for it.

Mobile hosts are computers with a wireless communication interface and with a system-wide unique identification. Each *Mh* may be either in the *active state* or *inactive state* (e.g. power save state, turned off, or simply unreachable), in which case it is unable to receive/send any messages from/to an *Mss*. We also assume that in order to become part of the system, an *Mh* must explicitly register itself, by sending a *join* message to the Mss of the cell where it is currently located. This *Mss* then becomes the new access point and the station responsible for the *Mh* (Mss_{Mh}). And accordingly, a *Mh* leaves the system by sending a *leave* message to its Mss_{Mh}.

Whenever an *Mh* enters a new cell it sends a greet message to the *Mss* responsible for the new cell, informing the identification of the *Mss* responsible for the cell it is leaving. With this information the *Mss* of the new cell is able to initiate a Hand-off protocol with the previous *Mss* with the purpose of transferring

some data related to the moving *Mh*. After completion of the Hand-off, the *Mss* of the new cell becomes the new Mss_{Mh}. The Hand-off protocol is described in more detail in section 3.4.

The greet message is also sent by the *Mh* when it becomes active again within the same cell where it was before becoming inactive. In this particular case, however, the *Mss* will not initiate a Hand-off protocol, since the *Mss* mentioned in the greet message is itself. In this model we abstract from the specific details of how a *Mh* learns that it is entering or leaving a cell and assume that this can be achieved in different ways according to the wireless technology being used.

Figure 1 shows a system with four cells served by four *Mss*s, two domains (with two cells each) managed by *IMS*1 and *IMS*2, and six *Mh*s. In the scenario, mobile host Mh_3 is requesting a multicast to the group (Mh_1,Mh_2). The figure also suggests that while the multicast is being processed, Mh_2 is migrating from cell 2 to cell 3. This scenario will be further explored in section 3.3.

Summarizing, the main assumptions of the model are the following:

1. Communication between any two *Mss*s is reliable, message delivery is in causal order, and *Mss*s do not fail;
2. Communication between an *Mss* and all the *Mh*s within its cell is unreliable, but there exists a maximum communication latency (λ) for wireless transmissions in all cells, and transmission faults (e.g. message losses) can be detected by any of the communicating partners.
3. At any time, each *Mh* in the system is associated with exactly one *Mss*, called its Mss_{Mh} (i.e. if the *Mh* is in a region of cell overlapping it must select one *Mss* as its Mss_{Mh}). An *Mh* must not reply to any message from any *Mss* other than its Mss_{Mh};
4. If an *Mh* is active it must send an acknowledgment for all messages received from Mss_{Mh}, and while it is inactive it must not reply to any message. *Mh*s may only leave the system after flushing all their pending message acknowledgments.

At this point we should make some comments on the degree of realism of the model. The reliability of the *Mss*s and the wired communication can be guaranteed by replication techniques [1] and well-known reliable communication protocols, and causal delivery is easily incorporated into wired distributed protocols. Concerning assumption 2, although wireless communication is inherently unreliable, disconnections or data loss due to interference can usually be detected "with reasonable accuracy" at the protocol layers up to the MAC layer, e.g. by listening to 802.11's MAC Beacon frames, reading management frame field *Status Code* [10], or by monitoring the RF signal or the *signal-to-noise ratio*. In most wireless networks, although a mobile host is able to receive signals from more than one base station, above the MAC layer it considers itself being served by (or connected to) a single *Mss*, which is exactly our assumption 3. Finally, in assumption 4, the requirement of flushing pending acknowledgments is only used to ensure the proper termination of the protocol, e.g. the eventual removal of information concerning the outcome of previous multicasts. It could be safely removed if we assume that the system does some periodic garbage-collection concerning such information.

3 The Protocols

This section is focused on the description of the iAM^2C protocol, but we start by giving an overview of the simpler AM^2C [9] and explaining our motivation for designing iAM^2C. As mentioned, the latter is a variant of the former, and solves the problem of scalability with respect to the set of Mobility Support Stations in the network.

3.1 The AM^2C Protocol

The AM^2C is a Two-Phase-Commit Protocol which essentially works as follows: whenever a new multicast is requested by an *Mh*, the corresponding Mss_{Mh} initiates the 2PC (hence, we will refer to it as Mss_{init}) by broadcasting the message to all other *Mss*s, which then relay the message to their local *Mh*s that are also in the multicast address list, and wait $T1$ time units for a positive (or negative) acknowledgment from each *Mh*. After this, each *Mss* computes the conjunction of all acknowledgments received, and replies with either OK or NOk to Mss_{init}. If an *Mh* addressed by the multicast migrates during this phase, then AM^2C executes a conservative (pessimistic) action to ensure atomic message delivery: it forces the *Mss* of the previous cell to generate a spontaneous NOk acknowledge message to Mss_{init} if the multicast message has already been forwarded to the migrating MH. This behavior obviously increases to the probability of aborting a multicast, specially for high migration rates. The first phase is completed as soon as Mss_{init} receives replies from ass *Mss*s.

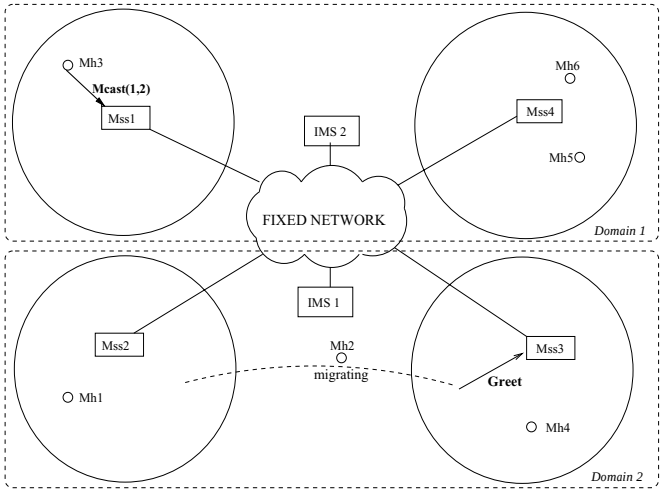

Figure 1: System with four *Mss*s, two *IMS*s and six *Mh*s

In the second phase, the Mss_{init} either commits or aborts the multicast, and again broadcasts this result to all *Mss*. These in turn forward this result to their local *Mh*s and wait for acknowledgments, which are sent back to Mss_{init}. However, unlike in the first phase, this final multicast status is re-forwarded to an *Mh* whenever it enters a cell, until all of the addressed *Mh*s have acknowledged its receipt. Only when all the destination *Mh*s have acknowledged the multicast status – and this has been checked at Mss_{init}– does the Mss_{init} notify all *Mss*s that they should discard this information.

The major problem of AM^2C is its lack of scalability regarding the number of *Mss*s, which has several facets: not only is the wired network overloaded with message broadcasts to the *Mss*s and their corresponding replies to the Mss_{init}, but also memory space and timeout processing is wasted at all *Mss*s which do not have any local *Mh*s in the list of the multicast destinations.

In order to overcome these limitations, we designed and implemented the *Indirect*AM^2C- iAM^2C), which employs a two-level hierarchical approach for keeping track of the locations (e.g. cells) of the *Mh*s.

3.2 Motivation

The main motivation underlying the design of iAM^2C was to constrain the sending of the multicast message only to those *Mss*s which actually have some local *Mh*s in the list of the multicast destinations (say, *M.Dest*). If there were no mobility of hosts this would be very easy, but since *Mh*s can migrate among *Mss*s, this locality information must be updated after each migration.

Since neither a centralized nor a fully distributed approach for keeping such location information seemed reasonable, we decided to introduce a new element called (*Intermediate Multicast Server* - *IMS*) which is responsible for controlling the delivery of multicasts to *Mh*s located in any cell within a *domain*. Our protocol assumes that every cell in the system is a member of exactly one domain (i.e. that every *Mss* has exactly one *IMS* associated to it), but that a domain includes several cells. Hence, supposedly the number of *IMS* is much smaller than the number of *Mss*s.

By keeping track of the location (i.e. cell) of each *Mh* within its domain the *IMS* is able to forward the multicast messages only to those *Mss*s that have access to the *Mh*s addressed in a multicast. Compared to AM^2C, this introduces a new level of indirection, since multicasts are first broadcast to all *IMS*s, then to some *Mss*s, and finally to the corresponding *Mh*s, but on the other hand, this makes the protocol scalable and saves resources at both the *Mss*s and wired network.

Yet another benefit is a clearer separation of tasks between the *IMS*s and the *Mss*s. While the former are now only responsible for storing the multicast messages (and their final status) and re-forwarding that information on demand, the latter elements are focused on executing the handoff protocol and handling the timeouts associated to non-responsive *Mh*s.

On the other hand, iAM^2C introduces some additional complexity related to the correct management of the mobile hosts' location, and to the re-forwarding of messages from *IMS*s to the *Mss*s, which becomes

necessary whenever an *Mh* migrates during the protocol execution.

3.3 The iAM^2C in More Detail

Like its predecessor, the iAM^2C is a Two-Phase -Commit Protocol with an additional third phase for clean-up. In the following, we describe how location management is done in iAM^2C, and then summarize the main events and actions of each phase.

In the following, let's assume that a multicast request M is issued by a Mh_s to all *Mh*s in the multicast's destination list *M.Dest*, which contains Mh_1 and Mh_2. Let's further assume that both Mh_1 and Mh_2 are initially within the cell controlled by Mss_2, which is associated with IMS_1.

In order to know to which *Mss*s it should forward a multicast message, each *IMS* keeps a *Location Map - LocMap*, indexed by the Mh_{ID} and containing the current cell of each *Mh* (i.e. the address of the controlling *Mss*). Whenever an *Mh* migrates between cells within a domain, the *Mss* of the new cell (e.g. Mss_n) sends the message LocUpdate (LU) to the *IMS* informing that Mss_n is now the new station responsible for *Mh*. When an *Mh* migrates between cells of different domains, then both the *Mss* of the new and the old cell (e.g. Mss_o and Mss_n) send LU messages to their corresponding *IMS*s, so that both are able to update their *LocMap*s. The sending of the LU messages is part of the Hand-off component of iAM^2C and will be discussed in more detail in section 3.4. The following is the behavior in each protocol phase:

Phase I This phase starts when an *Mss* responsible for Mh_s's cell receives a new multicast request M and assumes the role of the multicast initiator (or coordinator). This host, referred to as Mss_{init}, then sends M to each *IMS* in the system, which stores the message in a local buffer, and forwards M to some of its associated *Mss*s, according to the *Mh* –*Mss* association found in its *LocMap*. In our case, *IMS*1 would forward M only to Mss_2 since both Mh_1 and Mh_2 are within its cell.

Upon delivery of M each *Mss* forwards M to all *Mh*s $\in$ *local_Mhs* $\cap$ *M.Dest* and sets a local timeout of T1 time units, waiting for either a positive (OK) or negative (NOk) acknowledgment from the *Mh*s. As soon as either T1 expires or all local *Mh*s have replied, each *Mss* sends to Mss_{init} a single message OKSet containing the lists of positive and/or negative acknowledgments received from each *Mh*.

Mss_{init} keeps collecting those replies, and as soon as it has either an OK or NOk for every *Mh*$\in$ *M.Dest*, it decides about the final status of the multicast (committed or aborted), and the protocol enters Phase II.

Phase II This phase starts by having Mss_{init} send the final status to all *IMS*s (messages Comm or Abor), which forward this message to the corresponding *Mss*s, and these, in turn, to the corresponding *Mh*s $\in$ *local_Mhs* $\cap$ *M.Dest*. Upon delivery of this status message, each *IMS* removes M from its buffer, while keeping a record of the multicast's final status.

Like in Phase I, each *Mss* sets a local timeout of T2 time units, waits for Ack messages from the local *Mh*s, and finally replies to Mss_{init} with message AckSet, containing the list of *Mh*s that acknowledged.

As soon as Mss_{init} receives acknowledgments from all *Mh*$\in$ *M.Dest*, it knows that all *Mh*s have received the final status and will either deliver M to their applications (i.e. if status is committed), or not (i.e. if status is aborted).

Phase III Since the final status is of no more use, Mss_{init} sends message Del to all *IMS* so that these may remove this information from their buffers. Termination of iAM^2C is guaranteed by the assumption that no *Mh* leaves the system without explicit de-registration, and hence, no multicast will remain pending forever.

The major problem arises when any *Mh*$\in$*M.Dest* migrates during the execution of the protocol, since in this case the message M (or the status message) may have been forwarded to an *Mss* which is no longer responsible for the migrated *Mh*. Instead of simply aborting a multicast if any such *Mh* fails to send a OK/NOk acknowledgment before timeout T1 occurs at the *Mss* of the previous cell, iAM^2C makes an effort to reach the migrated *Mh* at its new cell, by retransmitting M to Mss_n.

For this to be possible, in addition to the interactions between Mss_o and Mss_n (like in the original AM^2C), the handoff component of iAM^2C also uses the LU message (from Mss_n to the *IMS*) as a request to re-forward to Mss_n all pending, non-delivered multicasts for the migrating *Mh*. To keep track of which messages from initiator Mss_i have been delivered (and acknowledged) by an *Mh*, in the protocol we maintain an array of lists called h_RECD, where

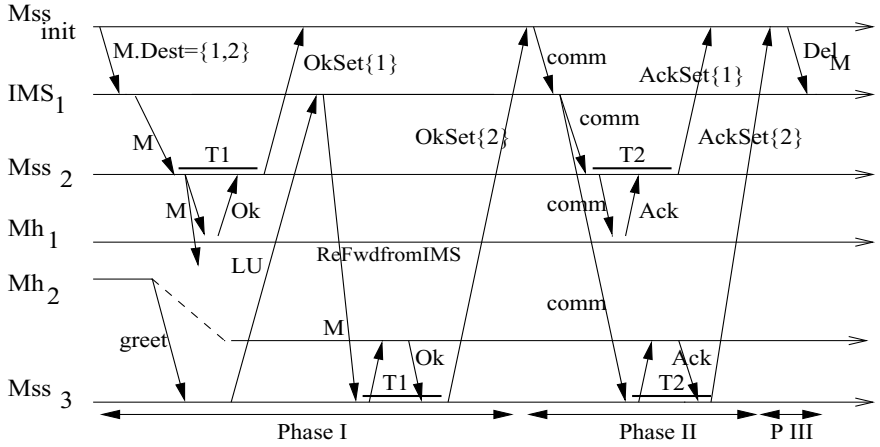

Figure 2: Execution of iAM^2C with migration of Mh_2

list $h\_RECD[i]$ stores all the recent message IDs (from Mss_i) that have been acknowledged by Mh.

Such a situation is shown in Figure 2, where Mh_2 migrates from Mss_2 to Mss_3 during Phase I. As soon as Mh_2 arrives at Mss_3 (for the sake of simplicity, we have omitted the hand-off messages exchanged between the Msss), this host sends a LU to IMS_1 both to update $LocMap$ and to request transmission of M, which is then delivered to Mh_2. Since in this scenario both Mhs have acknowledged before T1, Mss_2 and Mss_3 send messages OKSet{1} and OKSet{2}, respectively, to Mss_{init}. If this were not the case, no OKSet would have been sent. Assuming, however, that no new migrations occur in the remainder of the protocol execution, interactions in Phases II & III proceed as shown in Figure 2.

3.4 The Hand-off protocol

The Hand-off protocol is the part of iAM^2C which handles the mobility of hosts. It defines the interactions between Mss_o, Mss_n and their associated IMSs (say, IMS_o and IMS_n) that are triggered when a moving host (e.g. Mh_{new}) enters the cell of Mss_n (leaving Mss_o's cell). In the following, we will also assume that $Mh_{new} \in M.Dest$ for a given multicast M initiated by Mss_{init}.

As soon as Mh_{new} announces itself at Mss_n through message greet (informing the identity of Mss_o), Mss_n sends to Mss_o a dereg(Mss_n,Mh_{new}) message, announcing itself as the new access point for Mh_{new}.

When Mss_o receives dereg it removes Mh_{new} from $local\_Mhs$, and for all multicasts M for which $Mh_{new} \in M.Dest$ and it is still waiting for an acknowledgement from Mh_{new}, it replies with a NOk $_{new}$ to Mss_{init}. Then it includes M.ID into Mh_{new}'s h_RECD[Mss_{init}] and sends to Mss_n message deregAck containing the entire h_RECD array. Moreover, if Mss_n is not a member of IMS_o's domain, Mss_o further sends a LU message to IMS_o, asking it to remove Mh_{new} from its $LocMap$.

Upon reception of deregAck, Mss_n first will include Mh_{new} into its $local\_Mhs$ and then send message LU (containing Mh_{new}'s h_RECD) to IMS_n. With this message IMS_n not only updates its $LocMap$ (e.g. Mh_{new}'s new location), but also selects those multicast messages which are to be sent to Mss_n: i.e. those which are in IMS_n's buffer, and which are not mentioned in h_RECD as acknowledged. Notice that IMS_n has to do this selection for all possible initiators, i.e. it must examine each list h_RECD[i]. Then it packs all the selected messages into one message, ReFwdfromIMS, and sends it to IMS_n.

In the meantime, Mss_n may have received directly some of the messages within ReFwdfromIMS (e.g. because it has other $Mh_j \in local\_Mhs \cap M.Dest$), may have already sent them to Mh_{new}, received the acknowledgement and may have updated h_RECD accordingly. Therefore, when receiving ReFwdfromIMS, Mss_n will use h_RECD to select only those messages (within ReFwdfromIMS) that have not already been sent to Mh_{new} for re-transmission over the wireless link. After re-transmission, Mss_n sets the timer to T1 time units, and starts waiting for acknowledgements.

Regarding this re-transmission over the wireless link, we identified two possible policies: the *conditional re-transmission*, where a multicast message is never forwarded repeatedly by an Mss to its $local\_Mhs$; or the *unconditional re-transmission*, where a message

may be sent more than once over the wireless medium, provided there is some $Mh \in M.Dest$ that has not yet acknowledged its receipt. Although the first policy saves wireless bandwidth, the unconditional policy is more effective in reducing the probability of multicast aborts.

4 Complexity Analysis

In this section we analyze the message complexity of both protocols, and identify the scenario in which iAM^2C is more efficient than AM^2C.

The analysis considers the worst-case complexity (for a single multicast request) and distinguishes between *wired (Wd)* and *wireless (Wl)* messages. It assumes that the network is composed of I IMSs, N Msss and M Mhs, of which $D \subseteq M$ are destinations of the multicast. We also consider that all Mhs have a common migration rate mig[1] (number of hand-offs in some unit of time[2]), and use mig_M and mig_D to denote the rate mig when applied to the sets N and D, respectively. Hence, $mig_D \approx mig_M$ **card (D) / card (M).*

4.1 Complexity of AM^2C

The message complexity of AM^2C is given by the following equations:

$$\begin{aligned} Wd = \quad & (2*N + 2*mig_M*T1) + \\ & (2*N + 2*mig_M*T2 + mig_D*T2) + N \\ Wd = \quad & 5*N + 2*mig_M*(T1+T2) + mig_D*T2 \end{aligned} \quad (1)$$

$$\begin{aligned} Wl = \quad & (1 + 2*D) + (2*D + 2*mig_D*T2) \\ Wl = \quad & 4*D + 2*mig_D*T2 + 1 \end{aligned} \quad (2)$$

The number of wired messages (Wd) in AM^2C is composed of: a broadcast and the replies to/from all Msss and the hand-off messages dereg and deregAck during Phase I (first term), plus a broadcast and the replies to/from Msss, the hand-off messages, extra Acks (during Phase II), and Phase IIIs Del messages (second term), giving the sum shown in (1).

The number of wireless messages Wl is composed of: the multicast request from an Mh to Mss_{ini} and the forward&replies to/from Mhs $\in M.Dest$ during Phase I (first term), plus the regular and extra, migration-depenent, forward&replies to Mhs $\in M.Dest$ in Phase II, resulting in the sum shown in (2).

[1] Note that *mig* tipically depends on the size of cells and the migration profiles of the mobile users, and hence must be determined empirically for each network and its users.

[2] This unit of time must be the same as the one used for representing the protocol time outs.

4.2 Complexity of iAM^2C

The message complexity of iAM^2C is given by the following equations:

$$\begin{aligned} Wd = \quad & (I + 2*D + 2*mig_M*T1 + \\ & 2*mig_D*T1 + mig_D*T1) + \\ & (I + 2*D + 2*mig_M*T2 + \\ & 2*mig_D*T2 + 2*mig_D*T2) + I \\ Wd = \quad & 3I + 4*D + 2*mig_M*(T1+T2) + \\ & 3*mig_D*T1 + 4*mig_D*T2 \end{aligned} \quad (3)$$

$$\begin{aligned} Wl = \quad & (1 + 2*D) + (2*D + 2*mig_D*T2) \\ Wl = \quad & 4*D + 2*mig_D*T2 + 1 \end{aligned} \quad (4)$$

Wd in iAM^2C consists of: diffusion to all IMSs, forwards&replies to/from a subset of Msss (of size $\leq D$), the hand-off messages (first line), two LU messages (at worst, considering inter-domain migration) and one ReFwdfromIMS message within Phase I (second line), plus a similar number of messages within Phase II (lines 3 and 4). Notice that, compared to the Phase I, there is only an additional factor due to AckSet messages that eventually have to be re-sent to some Mss_{Mh} with migrating Mhs, regardless of any timeout. And finally, adding the Phase III Del messages to all IMSs results in the sum shown in (3).

Regarding the wireless messages Wl, iAM^2C has exactly the same number as AM^2C, since these messages are only delivered to Mhs $\in M.Dest$ and because through array h_RECD iAM^2C guarantees that no phase I message will ever be sent twice to a migrating Mh.

4.3 Discussion

Comparing equations (1) and (3) of Wl worst-case complexity, we can now identify in which scenario $Wd_i \leq Wd_a$, where Wd_i and Wd_a denote the number of wired messages of iAM^2C and AM^2C, respectively. For this, we should also consider the worst case, i.e. $D = N$, which happens if (a) there exist at least as many destination Mhs as Msss, **and** (b) the Mh distribution is so "unfortunate" that each Mss is serving at least one $Mh \in M.Dest$. On the other hand, because I is presumably much smaller than N, we can

safely neglect it. In this case, we have the following inequation[3]:

$$\begin{aligned} Wd_i = \;& 3I + 4D + HO + 3mig_D * T1 + 4mig_D * T2 \\ \approx \;& 4N + HO + 3mig_D * T1 + 4mig_D * T2 \leq \\ & 5N + HO + mig_D * T2 = Wd_a \qquad (5) \end{aligned}$$

iff

$$3 * mig_D * (T1 + T2) \leq N \qquad (6)$$

Hence, from expression (6) one can conclude that iAM^2C causes fewer wired messages than AM^2C, if N is sufficiently large when compared to the number of destination hosts that migrate during $T1 + T2$, or in other words, if the migration rate mig is sufficiently low. It also suggests that the efficiency of iAM^2C depends on a fine tuning of the time-outs T1 and T2.

5 Simulation

In order to get a deeper understanding of the details regarding the implementation of the protocols and to complement the complexity analysis of section 4 with average-case performance measurements and figures, we prototyped and simulated both protocols for different mobility scenarios. For this we used MobiCS, a flexible prototyping and simulation environment for distributed protocols in mobile networks. In this section we first give an overview of the simulation environment, then discuss the iAM^2C implementation, and finally present the simulation results that we obtained.

5.1 The MobiCS Simulation Environment

*Mobi*CS [7, 17] is a Java framework for prototyping and simulating distributed protocols in mobile networks. The framework provides means for modular programming of distributed protocols and features interchangeable simulation models which do not affect the implementation of the distributed protocols.

*Mobi*CS can emulate a mobile computing environment in either *deterministic mode* or *stochastic mode*. In the first mode the user is able to define simulation scripts, each of which describes a specific execution scenario (i.e. a network configuration and a particular pattern of events, such as a migration, a disconnection, etc.). By simulating the protocol for such a script, the developer is able to test if his/her protocol behaves "as expected" for the particular scenario. In the second mode protocols are tested in random scenarios that are defined by assigning a probabilistic event generator to some simulated network elements, such as mobile hosts or network links. For *Mh*s the randomly generated events are usually migrations, disconnections, re-connections, or in the specific case of our protocol, new multicast requests. In both modes, the user is able to inspect the states of each protocol component and network element.

The implementation of a distributed protocol in *Mobi*CS is done following the composite programming model described in [6]. In this model, a protocol is composed of *micro-protocols*, which are protocol parts implementing small and well-defined functionalities and which interact through events. Many authors agree that distributed protocols for mobile computing are most naturally described as the composition of the following three micro-protocols:

- **Wired** handles all messages exchanged in the fixed portion of the network;
- **Wireless** handles all messages exchanged through the wireless interface;
- **Hand-off** handles all (wired or wireless) messages related to a host migration.

Figure 3 shows the main interfaces defined and implemented for each micro-protocol of iAM^2C.

Each micro-protocol has an internal state and a set of *handlers*, i.e. operations that implement the actions to be executed at the occurrence of a specific event addressed to the micro-protocol. Events may be of two types: *messages*, which are received from other micro-protocols, and *timer-events*, which are scheduled by a micro-protocol to be triggered (by the simulation layer) after a specific period of simulated time. This organization was also used for our prototype implementation of iAM^2C in *Mobi*CS.

We first declared each protocol message and its attributes, then defined the corresponding handlers at the appropriate micro-protocols and finally declared the classes implementing these handlers.

5.2 Deterministic Simulations

The purpose of the deterministic simulation mode is to test a protocol in some controlled mobility scenarios, where it is easier to trace whether the protocol behaves correctly. Since it is impossible to simulate a protocol in all possible situations of protocol-external events

[3] We use HO to denote the handoff overhead common to both protocols, which is $2 * mig_M * (T1 + T2)$.

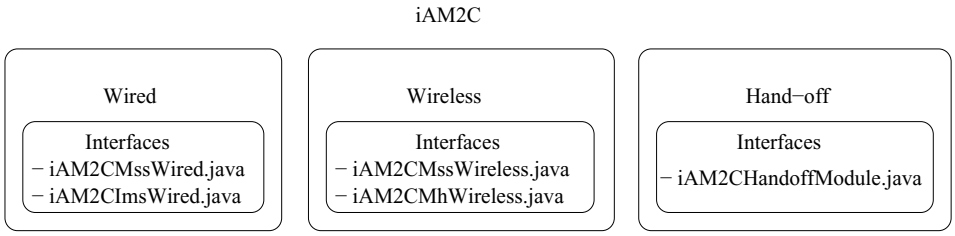

Figure 3: Interfaces of iAM^2C

(which would correspond to a protocol verification), only a few, more critical scenarios should be chosen. If the protocol executes correctly in these scenarios, the programmer at least can be sure that some behaviors of his/her protocol are correctly implemented.

The first steps for the deterministic simulations are always to decide which are the protocol behaviors to be tested in each scenario, and to choose the smallest set of network elements (e.g. *IMS*s, *Mss*s, *Mh*s) that is sufficient for all the scenarios. Then each scenario is defined, perhaps starting from a time-space diagram showing all the network elements and the pattern of events (e.g. migrations, multicast requests) which reflects the intended scenario. Finally, the pattern is coded into a *MobiCS script*. MobiCS provides several primitives for programming a script, such as setting global synchronization points, executing a migration, turning on/off message reception and timer events, and several others.

We simulated iAM^2C for several critical scenarios of concurrent migrations and multicasts, such as the one shown in Figure 2. In particular, we tested if iAM^2C executed correctly with inter- and intra-domain migrations (both in Phases I and II of the protocol execution) and for several wired message delivery orders, i.e. if multicast messages were delivered only once to migrating *Mh*s, if *LocMap*s were updated correctly, etc.

After our protocol implementation passed all the deterministic tests, we were ready to submit it to the random and hence more "chaotic" stochastic simulations, knowing that the protocol was correct at least for the scenarios tested in the deterministic mode.

5.3 Stochastic Simulations

With the stochastic simulations we wanted to evaluate iAM^2C's average-case performance when used in networks with different mobility ratios, and to compare it with the AM^2C protocol. Among others, we evaluated the total number of wired messages, the percentage of aborted multicasts, and the duration of a protocol execution (measured in *Simulated Time Units – STUs*) for several system configurations and for five migration probabilities $P_{mig} = \{0.01, 0.2, 0.4, 0.6, 0.8\}$.

For all simulations we used the following network configuration and simulation parameters:

- The static part of the network was composed of two *IMS*s, and different numbers of *Mss*s (from 8 to 30). We assumed full connectivity among the hosts, i.e. each fixed host was able to communicate directly with any other fixed host;
- The mobile part of the network consisted of 15 mobile hosts, all with a same migration probability P_{mig}, and same disconnection and reconnection probabilities ($P_{disc} = 0.01$ and $P_{reconn} = 0.3$)[4];
- The latency of wireless transmission was set to 50 times the wired transmission latency (which was set to one STU). The larger wireless latency was used to model the lower throughput and higher error rates of the wireless interface;
- Timeouts T1 and T2 were set to either 125 (or 175) STUs, which is somewhat more than 2 (or 3) times the wireless latency (wl_lat);
- All multicasts were addressed to all 15 *Mh*s and were requested with probability $P_{req} = 0.1$, after every 100 STUs;

We first measured the total number of wired messages of both protocols, i.e. Wd of section 4.

Figure 4 shows that already for eight *Mss*s (four per domain), iAM^2C uses many fewer (30%-50%) wired messages than AM^2C. This is probably due to the fact that because of the small number of cells, most *Mss*s are anyway already serving at least one *Mh*, such that most migrating *Mh*s can be served directly within

[4]We used probability P_X with linear distribution for deciding, after every 100 STUs, if the corresponding event would be generated.

the T1/T2 time-span, thus requiring less retransmissions from *IMS*s and less Ack messages. This result in fact shows that iAM^2C's average-case behavior is much better than its theoretical worst-case estimate presented in section 4.

In order to assess iAM^2C's scalability with respect to the number of *Mss*s, we compared the total number of wired messages of both protocols for different sets of *Mss*s. Figure 5 shows that for $P_{mig} = 0.2$ (which we consider a reasonable probability) the total number of wired messages is more or less constant in iAM^2C, while in AM^2C it keeps increasing with the number of *Mss*s, as expected.

Then we compared both protocols regarding the percentage of multicast aborts and tried to identify how dependent on the migration probability this percentage is. As can be seen from Figure 6 the ratio of aborted multicasts is similar in both protocols, but the iAM^2C values are always a bit smaller. Because both protocols essentially adopt a *pessimistic approach* (see section 3), the ratio of aborts grows very fast with the increase of P_{mig}, and is not much influenced by the value of timeout T1 (as long as it is large enough for allowing a request-reply interaction over the wireless interface). The lower abort ratio for iAM^2C was obtained because in this protocol an Mss_o only replies with NOk for a migrating *Mh* if the dereg message arrives after timeout T1.

We also measured the amount of simulated time that each of the two protocols takes to complete. Figure 7 clearly shows the influence of iAM^2C's additional level of indirection (i.e. path through *IMS*s) on the total execution time, and results show more than twice the time required for AM^2C. Thus, in our opinion, this is the main drawback of iAM^2C when compared to AM^2C.

As expected, the choice of time-outs $T1$ and $T2$ is crucial for the proper functioning of both protocols: if they are too close to the round-trip time of a request-reply over the wireless interface ($2*wl\_lat$), then nearly all multicasts are aborted. However, if they are much larger than $2*wl\_lat$, then on one hand both protocols become equally less sensitive to migrations (e.g. nearly all multicasts are committed); on the other hand, this causes an increase of the total execution time of both protocols, but especially of iAM^2C.

After many experiments, we found out that an appropriate value for the time-outs should be in the range $[2.5*wl\_lat, 3.5*wl\_lat]$, and that testing the protocols for other values did not give any interesting insights into the protocol's behavior. Apparently, the aforementioned interval defines a good trade-off between accuracy of non-reachability detection and protocol duration.

6 Related Work

We are unaware of other work which uses hierarchical location management within a 2PC for implementing a scalable *reliable multicast*.

Hierarchical Location Management has been thoroughly researched, especially for Mobile Computing. Pitoura and Samaras [14] give a broad survey and classification of approaches. According to their taxonomy, the iAM^2C uses a combination of level caching and replication for maintaining the information concerning the locality of *Mh* and for delivering the messages and final status of multicasts. In fact, the *IMS* manages a cache with location information of certain *Mh*s (i.e. those located in its domain) which allows the protocol to narrow the set of addressed *Mss*s. And like most cached information, it may also become inconsistent when a *Mh* migrates, and hence has to be updated. The *IMS*s also act as repositories of replicas of multicast messages and final status, which can be re-forwarded to the appropriate *Mss* on demand.

Concerning iAM^2C's delivery semantics, there are very few protocols which guarantee *all-or-nothing* multicast message delivery. In most other work the multicast protocols provide a weaker delivery guarantee, e.g. that messages will eventually be delivered to all the destinations (*Mh*s), but where the addressees do not get a feedback whether the multicast was or not accepted by all of them.

Acharya and Badrinath's MCAST[2] protocol guarantees reliable multicast with *exactly once* delivery. From their work, we borrowed the idea of using the *IMS*s as intermediate repositories for pending multicasts.

Prakash et al. present an efficient causal ordering algorithm [15], which can be easily combined with MCAST to enforce reliable, causally ordered multicasts in mobile systems. The major addition is that each wired and wireless message carries information about its direct predecessor messages with respect to each destination host. Hence, in a similar way their algorithm could also be incorporated into iAM^2C to implement causally ordered, reliable multicasts.

Algar and Venkatesan have also proposed three protocols for reliable and causally ordered message delivery in mobile computing systems [3] which could be used for reliable multicast. The three algorithms are extensions of Raynal et al.'s algorithm [16] and differ in the message complexity of the Hand-Off component and the size of the message headers (i.e. which

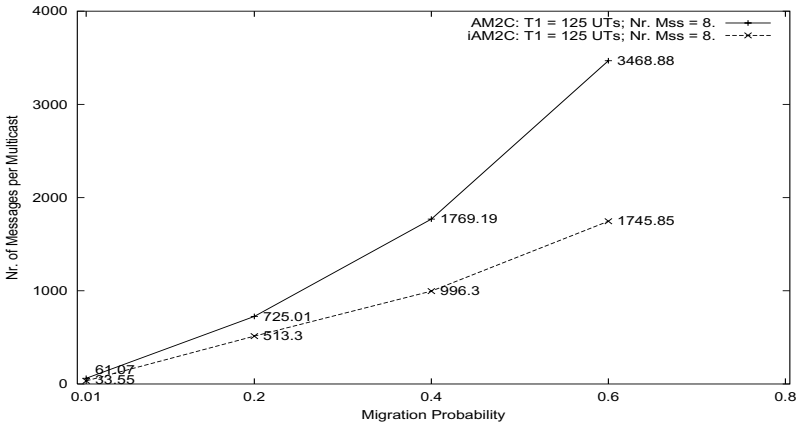

Figure 4: Total Wired Messages vs Migration Probability

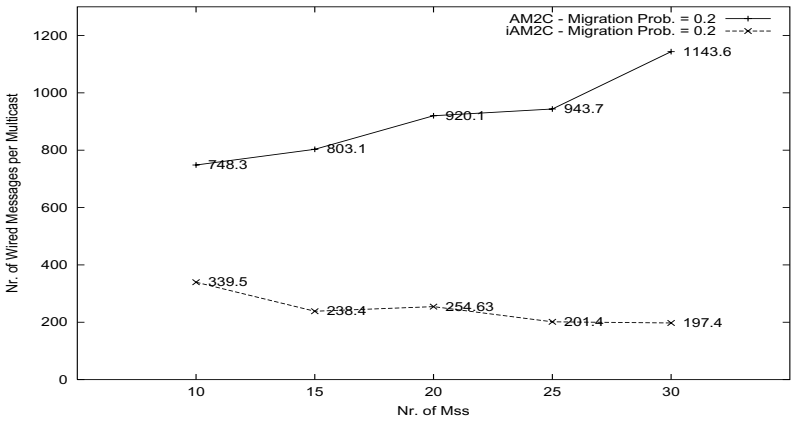

Figure 5: Wired Messages for different number of *Mss*s

carry information about causal dependencies). Unlike Prakash's algorithm, here most of the message ordering and filtering task is performed at the *Mss*s, which behave as the representatives of their *local Mh*s. Among the proposed algorithms, only the first one is similar to MCAST (and to AM^2C), while the other two are based on broadcasting causality information among the *Mss*s at every *Mh* migration, which obviously does not scale with respect to the migration probability.

Anastasi et al. have proposed a reliable multicast protocol with dynamic group membership and several ordering semantics [4], which also uses *Mss* as the intermediates for caching and relaying messages to the mobile hosts. Unlike the other approaches, their protocol lacks a *Hand-Off* component, which makes the protocol efficient in scenarios of high migration rates, and with many *Mh*s. However, the protocol requires *Mss* to periodically re-broadcast pending multicasts and, add sequence number information in the beacon messages, and *Mhs* to explicitly request retransmission of "missed" messages through *Negative Acks (NACK)* messages. This not only wastes the wireless resources and requires the wireless technology to support piggy-backing data on beacons (which is not always possible), but also requires more processing from the *Mh*s, which have to keep track of missing messages and send NACKs whenever necessary. A similar approach for reliable multicast using *NACK*s is presented in [5].

Some other work [18, 12, 11] extends traditional IP multicast protocols for mobile hosts, most of them relying on Mobile IP[13]. Rather than providing reliable multicast, these approaches guarantee only *best-effort*

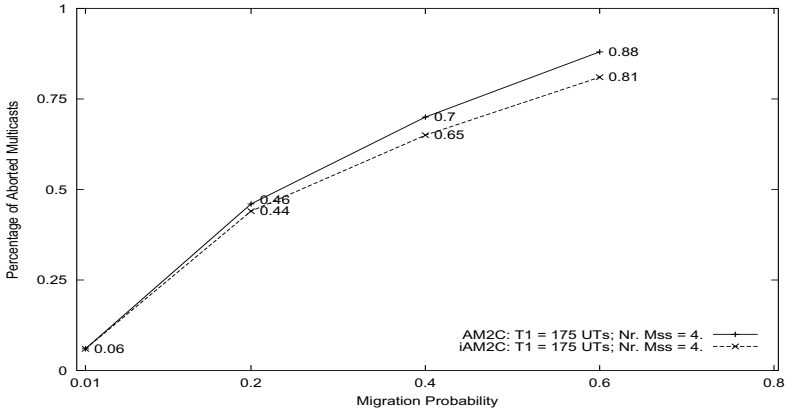

Figure 6: Percentage of Aborts vs Migration Probability

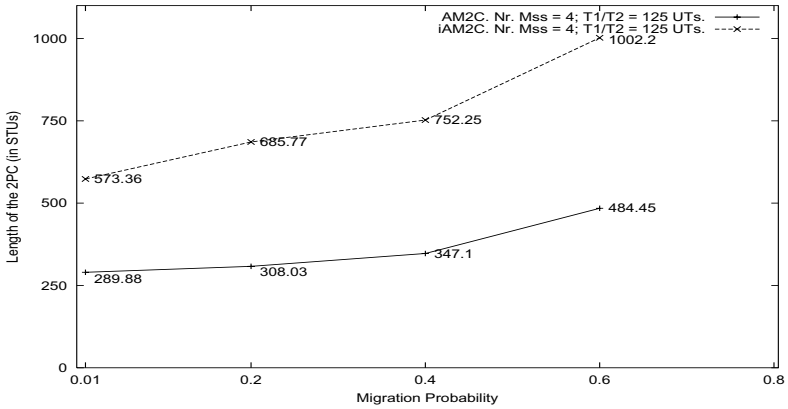

Figure 7: Length of the Protocol vs Migration Probability

multicasting and suffer from the scalability problems of Mobile IP.

7 Conclusion

In this paper we compared two protocols for reliable multicast in mobile structured networks, which we have prototyped, simulated and evaluated using the *Mobi*CS environment. In iAM^2C, we used a two-level hierarchical location scheme to constrain the set of *Mss*s that must participate in the protocol: at the higher level, the *IMS*s keep track of the *Mh*s located in their domain, and at the lower level, each *Mss* keeps track of the *Mh*s within its cell.

With this approach, it is possible to save resources of the wired network (bandwidth) and at the *Mss*s (memory). Another advantage is that retransmissions can be handled in a decentralized way by each *IMS*, which makes the protocol more scalable in the number of *Mss*s. The *Mss*s can also be much simpler, since all message storage and delivery control is done at the *IMS*s.

However, in order to keep this location information up-to-date at the *IMS*s, it was necessary to introduce an additional level of indirection in the communication path, and also to include additional wired messages (e.g. location updates and re-forwardings) in order to maintain location information consistent and to guarantee uniform message delivery, also to migrating hosts. While the former had a significant impact on protocol duration, the latter did not cause a noticeable increase in the number of wired messages. A problem that has not been addressed in our work is

that of efficient data structures for the *LocMap*, i.e. how an *IMS* can efficiently manage location information for a large number of *Mh*s.

Despite these problems, we think that iAM^2C is an improvement over AM^2C. Firstly, it provides a means for solving the scalability problem of reliable multicast with respect to the size of the static network. Secondly, it uses a less pessimistic approach to handle migrations, because it does not require the Mss_o to abort the multicast if it receives a dereg for a migrating *Mh* before the time-out in Phase I.

We are aware that neither iAM^2C nor AM^2C are scalable with respect to the *Mh*s addressed by a multicast, but this is an intrinsic problem of protocols for reliable multicast. And source controlled multicasts are in fact the best choice, since their complexity is only Θ(N). Moreover, we also believe that most applications for coordination and collaboration usually involve only a small number of users, rather than hundreds or thousands os them.

As future work we plan to develop multicast and group services with less strict delivery semantics and which incorporate appropriate handling of intermittent and frequent disconnections of hosts in a group.

Acknowledgments

Special thanks go to Ricardo C.A. da Rocha, who provided excelent support for the *Mobi*CS framework, and helped to solve our problems related to the use of this environment.

References

[1] S. Aalgar, R. Rajagopalan, and S. Venkatesan. Tolerating Mobile Support Station failures. In *Proceedings of 1st Conference on Fault Tolerant Systems, Madras, India*, pages 225–231, 1995.

[2] A. Acharya and B.R. Badrinath. Delivering Multicast Messages in Networks with Mobile Hosts. In *Proc. of 13th International Conference on Distributed Computing Systems, Pittsburgh*. IEEE Computer Society, May 1993.

[3] S. Alagar and S. Venkatesan. Causal ordering in distributed mobile systems. *IEEE Transactions on Computers*, 46(3) pages 353—361, 1997.

[4] Giuseppe Anastasi, Alberto Bartoli, and Francesco Spadoni. A reliable multicast protocol for distributed mobile systems: Design and evaluation. *IEEE Transactions on Parallel an Distributed Systems, 12(10), pages 1009-1022, October 2001.*

[5] V. Aravamudhan, K. Ratnam, and S. Rangajaran. An Efficient Multicast Protocol for PCS Networks. *ACM/Baltzer Mobile Networks and Applications (MONET)*, 6(2) pages 333–344, 1997.

[6] Nina T. Bhatti and Richard D. Schlichting. Configurable communication protocols for mobile computing. In *Proceedings of the 4th International Symposium on Autonomous Decentralized Systems*, pages 220–227, Tokyo, March 1999.

[7] Ricardo C.A. da Rocha and Markus Endler. MobiCS: An Environment for Prototyping and Simulating Distributed Protocols for Mobile Networks. In *Proc. 3rd IEEE International Conference on Mobile and Wireless Communications Networks (NWCN2001), Recife - Brazil, pages 44–51, August 2001.*

[8] Mateus de F. Ribeiro. Desenvolvimento e comparação de dois protocolos para multicast atômico em computação móvel. Master's thesis, IME/Universidade de São Paulo, Rua do Matão 1010, ZIP 05508-990, São Paulo, September, 2001.

[9] M. Endler. A protocol for atomic multicast among mobile hosts. In Proc. *Dial M Workshop/Mobicon'99, Seatle (USA)*, pages 56–63. ACM, August 1999.

[10] Jim Geier. *Wireless LANs*. SAMS, 2002.

[11] M. Handy, H. Schulzrinne, E. Schooler, and J. Rosenberg. Sip: Session initiation protocol. RFC 2543, Columbia University, March 1999. www.cs.columbia.edu/~hgs/sip/papers.html.

[12] T.G. Harrison, C.L. Williamson, W.L Mackrell, and R.B. Bunt. Mobile Multicast (MoM) Protocol: Multicast Support for Mobile Hosts. In *Proc. 3rd International Conference on Mobile Computing and Networking (Mobicom 97), Budapest, Hungary*, pages 151–160, September 1997.

[13] C.E. Perkins (editor). IP Mobility Support. RFC 2002, IBM, October 1996.

[14] E. Pitoura and G. Samaras. *Data Management for Mobile Computing*. Kluver Academic Press, 1998.

[15] R. Prakash, M. Raynal, and M. Singhal. An Adaptive Causal Ordering Algorithm Suited to

Mobile Computing Environments. *Journal of Parallel and Distributed Computing*, pages 190–204, March 1997.

[16] M. Raynal, A. Schiper, and S. Toueg. Causal Ordering Abstraction and a Simple Way to Implement it. *Information Processing Letters*, 39(6) pages 343—350, 1991.

[17] Ricardo C.A. Rocha. Mobics home page. http://www.lcpd.ime.usp.br/~mobics/. (Last visited on February 2003).

[18] George Xylomenos and George C. Polyzos. Ip multicast for mobile hosts. *IEEE Communications Magazine*, 35(1) pages 54–58, 1997.