Design and Evaluation of a Protocol for Atomic Multicast among Mobile Hosts

Mateus de Freitas Ribeiro[*], **Markus Endler**

Instituto de Matemática e Estatística - IME - USP
Rua do Matão, 1010
05508-090, São Paulo - Brazil
mateus@ime.usp.br

Departamento de Informática - PUC-Rio
Marquês de São Vicente 225 - Gávea
22452, Rio de Janeiro - Brazil
endler@inf.puc-rio.br

Abstract

In this work we extended Acharya and Badrinath's [2] reliable multicast protocol for mobile computing so as to provide also for atomicity of message delivery. In this protocol, named AM^2C, a multicast message is only delivered to all mobile host destinations if these are reachable during a first phase of the protocol. After a multicast is either committed or aborted, the protocol guarantees that all the mobile hosts are eventually informed about the final status of the multicast. In this paper we discuss the protocol and present results of our simulations.

Keywords: Mobile Computing, Atomic Multicast, Simulation

Resumo

Neste trabalho é proposta uma extensão do protocolo de multicast para computadores móveis definido por Acharya and Badrinath's[2] no sentido de prover também a entrega atômica de mensagens. Em nosso protocolo, denominado AM^2C, uma mensagem de multicast só é entregue a todos os computadores móveis se estes podem ser alcançados na primeira fase do protocolo. Depois que um multicast é ou confirmado ou abortado, o protocolo garante que todos os computadores móveis mais cêdo ou mais tarde são informados sobre o resultado final do multicast. Neste artigo discutimos e apresentamos o protocolo e os resultados de nossas simulações.

Keywords: Computação Móvel, Multicast Atômico, Simulação

[*]The author is currently at the Dedalus Sistemas, São Paulo, E-mail: mateus.ribeiro@dedalus.com.br

1 Introduction

With the ongoing improvement and spread of cellular telephony technology, more and more computer network services will be supporting wireless and mobile communication. In near future, it will be possible to use a huge variety of services from lightweight, inexpensive and hand-held terminals or PDAs, regardless of their current location. Some of these services will require some form of coordination or synchronization among groups of mobile clients. Examples of such services are strategic mission planing and execution, optimization in vehicle movement, cooperative design, 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 that multicast communication be atomic, i.e. that a multicast message is either accepted by all mobile hosts of the group or by none of them. This may be important for applications with requirements for strong consistency of the mobile user's view of the global system state.

In the following we present three scenarios/applications with such a strong consistency requirement.

Group of Traveling Sales-persons A group of traveling sales-persons need a coherent view of distributed data stored in the mobile computers of the group members. For example, assume that each of the sales-person's mobile computer stores a list of products (with number of items) that he/she is authorized to sell directly (e.g. without any query to the others), and that the total number of currently available product items for sale is exactly the sum of the number of items stored in all the group's laptops. Therefore, if one of the sales-person wants to sell a quantity that exceeds the local number of items, he/she will have to reserve the remaining items from some other sales-person in the group. And this request must be communicated and agreed upon by all group members since otherwise two or more concurrent sales (or bookings) may leave the distributed state (i.e. the total number of items) inconsistent.

Choosing a Movie A group of friends plan to go to a theater together and need to reach an agreement on which movie to watch, and at which time to go, since they all want to buy their tickets in advance from their mobile devices. So each of them in turn makes a proposal, and eventually the group reaches an agreement. Atomic multicast is of great value for such distributed decision-making, since alter each proposal (a multicast), each of them learns whether the group has or not reached an agreement. Moreover, if for some reason one of the friends is not reachable during a (hopefully short) period of time, then no proposal submitted during this period of time should be committed, since all friends want everyone in the group to meet.

Coordinated Position Tracking Members of an 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 sensoring through GPS) and 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 propose a protocol for *atomic multicast* among mobile hosts, which is an extension of Acharya and Badrinath's MCAST[2] reliable multicast protocol. In this article we focus on the property of atomicity (*all-or-nothing*), and do not discuss any ordering requirements. FIFO or causal message delivery could be incorporated into our protocol exactly in the same way as other work[14] which extended MCAST.

Essentially, our protocol is a Two Phase Commit (with an additional garbage-collection phase) executed among the mobile service stations, which are the intermediates in the communication between any mobile hosts. According to our protocol, a multicast message is only delivered to all mobile host destinations if these are reachable during the first phase of the protocol. In this case the multicast is *committed*, otherwise it is *aborted*. The second phase of the protocol guarantees that all the mobile hosts are eventually informed of the final status of the multicast. This is achieved by retransmitting this information to migrating mobile hosts until all addressee acknowledge its receipt. After this, the third phase performs the removal of the information about the multicast's final status.

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 proposed multicast protocol. Section 4 explains the Hand-off part of the protocol in more detail. In Section 5 we present results of the protocol simulation. Section 6 mentions related work and finally, in section 7 we draw some conclusions and mention our future work.

2 System Model and Assumptions

The model of the system has two types of machines, the *Mobile Service Stations (*Mss*)* and the *Mobile Hosts (*Mh*)*. The former are assumed not to fail, and are linked with each other through a static, reliable computer network. Each *Mss* further defines a geographic region, called *cell*, where it is able to communicate reliably with a set of mobile hosts currently located in the cell. The information about the *Mh*s within a cell is maintained at each *Mss* in the data structure *local_Mhs*. Each *Mss* also maintains the status (e.g. *outcome*) of each pending multicast request within in the system.

The mobile hosts are disconnected computers that have a system-wide unique identification. From the perspective of the remaining system, a *Mh* may be in two possible states: *activated* and *deactivated*. In the inactive state (e.g. power save state, turned off, or simply, unreachable) a *Mh* is unable to receive or send any message. A *Mh* joins the network by sending a *join* message to the *Mss* in charge of the cell it is currently in, which then becomes the *Mss* currently responsible for the *Mh*. (*respMss*). A *Mh* leaves the system by sending a *leave* message to any reachable *Mss*, and this message is broadcasted to all other *Mss*s.

Mobile hosts are able to move from one cell to another. Whenever a *Mh* enters a new cell it sends a *greet* (Mss_o) message to the *Mss* responsible for the new cell, and argument Mss_o is the identity of its former *respMss* (i.e. 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 *respMss* to transfer some data related to the moving *Mh* to the new *Mss*. After the Hand-off is completed, the *Mss* of the new cell officially becomes the new *Mh*'s *respMss*. The Hand-off protocol is described in more detail in section 4.

The *greet* message is also sent by the *Mh* when it becomes active again within the same cell where it has been before becoming inactive. In this particular case, however, the *Mss* will not initiate a Hand-off protocol, since the *Mss* mentioned in message *greet* 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 three cells (*Mss*s) and four *Mh*s, where mobile host Mh_1 is requesting a multicast to the group (Mh_1, Mh_2 and Mh_3). 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.

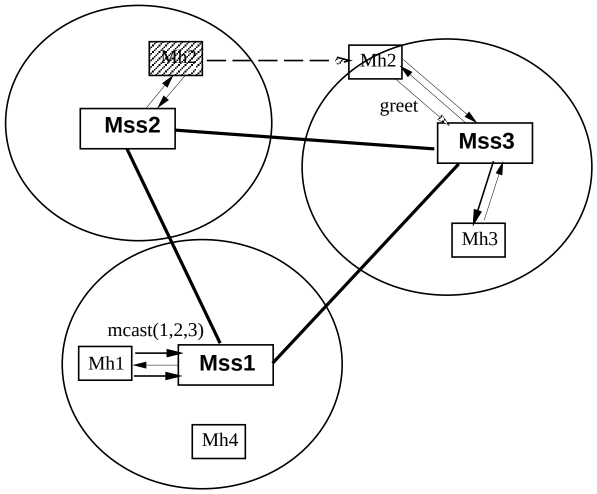

Figure 1: System with three *Mss* and five *Mh*s

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

1. Communication among any two *Mss*s is reliable, message delivery is in causal order, and *Mss*s do not fail;

2. Broadcasts in the static network (among fixed hosts, including *Mss*s) have an known upper bound on transmission time;

3. Communication among a *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.

4. At any time, each *Mh* in the system is associated with exactly one *Mss*, called its *respMss* (i.e. if the *Mh* is in a region of cell overlapping it must select one *Mss* as its *respMss*). A *Mh* must not reply to any message from any *Mss* other than its *respMss*;

5. If a *Mh* is active it must send an acknowledgment for all messages received from *respMss* and while it is inactive it must not reply to any message. But a *Mh* can only leave the system after flushing all its pending message acknowledgements.

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. Assumption 2 can be fulfilled either if the static network is a local network, or if it is a dedicated network with known traffic patterns. Concerning assumption 3, although wireless communication is inherently unreliable, disconnection or interference can usually and detected by both communication parties by measuring the RF signal and the *signal-to-noise ratio*. In most wireless networks, despite 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 4. Finally, assumption 5 defines the terms *active* and *inactive*, and the requirement of flushing pending acknowledgements is just used to ensure the proper termination of the protocol, e.g. the eventual removal of information concerning the outcome of previous multicasts.

3 Outline of the Protocol

The Atomic Multicast Protocol for Mobile Computing AM^2C is a Two-Phase Commit Protocol (2PC) with an additional phase for garbage collection. The protocol uses the *Mss*s as intermediate repositories of messages and gateways to the wireless links. In AM^2C a multicast is only committed if all the addressed mobile hosts accept the message within a certain time frame (first phase), otherwise it is aborted. The outcome of the protocol however remains stored at the *Mss*s and is re-forwarded to all addressee until all of them acknowledge the receipt of the outcome (second phase). Only then, is the outcome removed from all the *Mss*s (third phase).

In the following, we describe the protocol in more detail.

Phase I

1. A mobile host (Mh_{sender}) sends a new multicast request M to its *respMss*, which we will call Mss_{ini}. Each multicast has a unique identifier, which is composed of the identifier of Mss_{ini} and a sequence number *sn* (assigned by Mss$_{ini}$). Also as part of message M, a field *M.Dest* specifies the set of destinations (mobile hosts) of the message.

2. Mss_{ini} broadcasts the request M to all other *Mss*s and waits for replies from them.

3. When receiving a M from Mss_{ini}, all *Mss* in the system forward the request to *local_Mhs* $\cap$ *M.Dest* and wait for T1 units of time for replies from these *Mh*s. If either *local_Mhs* $\cap$ *M.Dest* is the empty set, or all local *Mh*s answer with Ok_M within T1 time units, then *Mss* sends an Ok_M to Mss_{ini}. Otherwise it sends NOk_M to Mss_{ini}. This is also done by Mss_{ini}, but the only difference is that Mh_{sender} is not included in the set of local Mh destinations.

4. When any *Mh* receives M from its *respMss*, it stores M in a local buffer, and immediately replies with either a Ok_M or NOk_M, depending on whether the application program at the *Mh* is willing or able to accept M.

5. When the M message arrives at *Mss* during the migration of a new *Mh* (say Mh_{new}) into *Mss*'s cell, then depending on the moment of Hand-Off completion, the waiting time is extended by another T1 time units, in order to allow for the forwarding of the message M also to Mh_{new}. This behavior is discussed in section 4 in more detail.

6. If a *Mh* resumes activity in its previous cell, then its *respMss* re-forwards to *Mh* all the multicasts M for which timeiut T1 has not yet occurred.

Phase II

1. After receiving replies from all *Mss*s, Mss_{ini} checks if all replies are of type Ok_M or if any of them is of type NOk_M. In the first case it sets the final status of the multicast to *Commit* and in the latter case to *Abort*. It then stores either $Comm_M$ or $Abor_M$ in its log *outcomes* and broadcasts this status to all *Mss*s.

2. When receiving either an $Abor_M$ or a $Comm_M$ message from Mss_{ini}, each *Mss* stores this message in its log *outcomes*[1], forwards it to *local_Mhs* $\cap$ *M.Dest* and waits a period T2 of time for an acknowledgment from these local *Mhs*. Let C be the set of local *Mhs* that acknowledged. *Mss* then sends message Ack_M C back to Mss_{ini}.

3. After a new Mh, say Mh_{new}, registers with a *Mss*, the *Mss* checks if $Mh_{new} \in M.Dest$ for any M $\in$ *outcomes*. For every such message, *Mss* forwards the final status of message M to Mh_{new} and waits at most time T2 for an acknowledgment Ack_M. If this acknowledgment arrives in time, Mss sends an Ack_M $\{Mh_{new}\}$ to Mss_{ini}, and otherwise it does not send an Ack_M message. For every M $\notin$ *outcomes* or $Mh_{new} \notin M.Dest$, the arrival of Mh_{new} at *Mss* has no influence on the result Ok_M or NOk_M.

Phase III

1. Each time Mss_{ini} receives a message Ack_M C from a *Mss*, it adds the elements in C to a set *M.Replied*. When eventually *M.Replied* = *M.Dest* then Mss_{ini} removes the corresponding entry from log *outcomes* and broadcasts message Del_M to all *Mss*s.

2. When receiving Del_M from Mss_{ini}, each *Mss* removes M from its own log *outcomes*

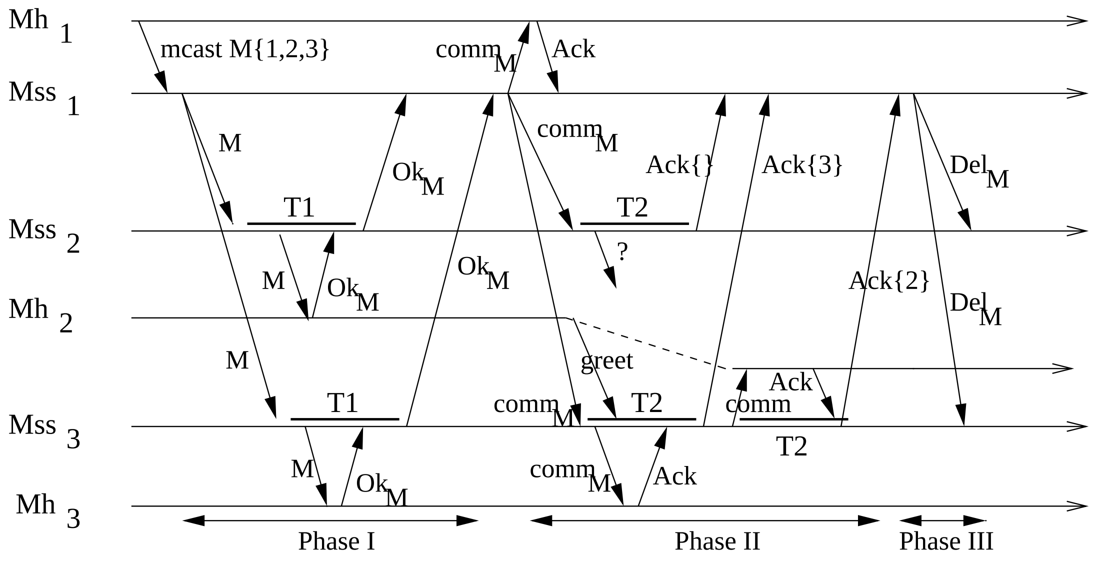

Figure 2: The Multicast Protocol

Figure 2 shows the types of messages transmitted in the protocol in the following scenario: Mh_1 sends a multicast M to mobile hosts Mh_2 and Mh_3, and Mh_2 migrates from Mss_2 to Mss_3 in between the first and the second phases. Notice that all *Mss*s keep

[1] *Mss* also removes the corresponding multicast message M from its buffer.

the result of a multicast request in their logs *outcome* until all *Mh*s have acknowledged the receipt of this result (e.g. Mh_2 sends this acknowledgment only when it is within Mss_3's cell). By this, we guarantee non-uniform consistency[7] of message delivery to the mobile hosts, meaning that all *Mh*s which are registered and eventually become active for a sufficiently long period of time will be informed of the final status of each multicast M, for which $Mh \in M.Dest$.

4 The Hand-Off Protocol

The Hand-off protocol defines the interactions between two *Mss*s when a *Mh* moves from the old cell (serviced by Mss_o) to the new cell (serviced by Mss_n). In the following, we describe these interactions for a migrating *Mh*, (Mh_new) which is also member of *M.Dest* for a given multicast M.

1. After Mh_new registers with a Mss_n (by sending a *greet* message), this *Mh* is included in a (temporary) list *future_local_Mhs*. Mss_n then sends a *dereg(*Mss$_n$*,Mh_new)* message to Mss_o and waits for the corresponing acknowledgment, through message *deregAck*.

2. When Mss_o receives *dereg(*Mss$_n$*,Mh_new)*, it immediately removes Mh_{new} from *local_Mhs*, replies (to the corresponding Mss_{ini}) with NOk_M for all messages M s.th. $Mh\_new \in M.Dest$ and for which it is still waiting for either a Ok_M or NOk_M from Mh_{new}. Then, it also sends message *deregAck* to Mss_n.

3. When the message *deregAck* from Mss_o arrives at Mss_n responsibility for Mh_{new} is officially established at Mss_n (i.e. Mh_{new} is added to *local_Mhs*). Message *deregAck* carries an array (*h_RECD*) similar to the one described in [2]. Each element *h_RECD[i]* holds the largest sequence number *sn* of a multicast initiated at each Mss_i and replied by Mh_new. Number *h_RECD[i]* also means that Mss_o has handled the replies (to Mss_i) for all multicasts up to this sequence number (and greater than the corrsponding sequence number received when Mh_new entered Mss_o's cell).

4. Based on the sequence numbers in *h_RECD*, Mss_n decides what to do for buffered multicast messages M s.th. $Mh_{new} \in M.Dest$, and for which Mss_n is still waiting for replies from the local *Mh*s $\in M.Dest$. Essentially, there are two possible actions depending on whether Mss_o has or not already forwarded M to Mh_{new} (and replied to Mss_{ini}): if it did already handle the reply, then after timeout T1 Mss_n replies to Mss_{ini} without considering Mh_{new} (e.g. as if its contribution to the conjunction of *Mh* replies were Ok_M); otherwise Mss_n re-forwards M to Mh_{new} and extends the waiting time for Mh_{new}'s reply by another T1 time units. However, for all multicasts M with $Mh_{new} \in M.Dest$ for which its timeout T1 expires before arrival of *deregAck* (and if *h_RECD* indicates that Mss_o has not yet handled the reply), Mss_n replies with NOk_M.

The desired atomicity property of AM^2C requires that a multicast is to be commited only if it can be guaranteed that indeed all the *Mh*s in *M.Dest* received and accepted the message. Therefore, if there is any uncertainty about this fact, then the multicast must be aborted.

The Hand-Off protocol described above ensures this behavior: Until completion of the Hand-off Mss_o remains responsible for sending the replies (to any possible Mss_{ini}) for all multicasts messages forwarded to the local *Mh*s. Through ith element of array *h_RECD* Mss_o also informs Mss_n of the latest multicast for which it already replied to Mss_i. Since wired message delivery times are unpredictable, Mss_o can not be sure that Mss_n will receive message *deregAck* early enough in order to be able to re-forward M to Mh_{new}, and hence Mss_o must reply to Mss_{ini} with NOk_M, if Mh_{new} did not yet reply. On the other hand, Mss_n must reply to Mss_{ini} with an extra NOk_M message if *dregAck* arrives to late (e.g. timeout T1 for *Mh* replies has already occurred), and it learns that Mss_o did not handle the reply for this multicast (i.e. it probably replied considering Ok_M for Mh_{new}).

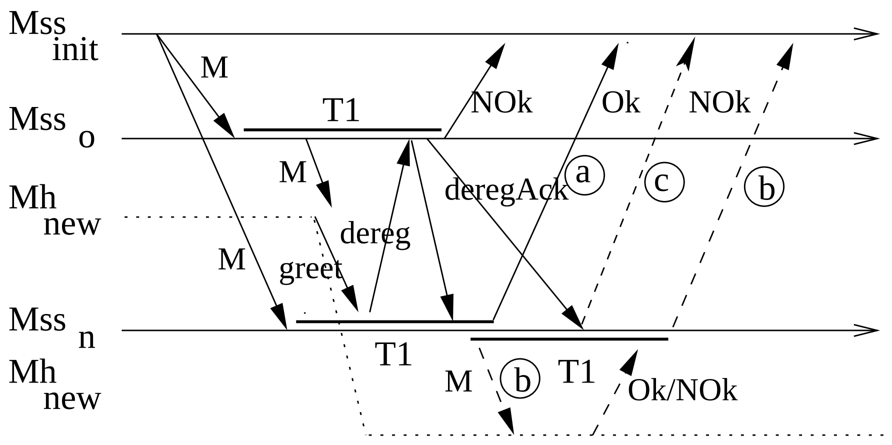

Figure 3: The Hand-off problem

Figure 3 shows an example in which a migrating Mh_{new} does not send a reply to Mss_o and the *dereg* message arrives at Mss_o before $New{+}T_1$, where *New* denotes the moment a new multicast message M arrives at each *Mss*. In this case Mss_o replies to Mss_{ini} with NOk_M and informs Mss_n (through *h_RECD* in *deregAck*) that it has already handled the reply to Mss_{ini}. At Mss_n, *greet* arrives after *New* but before $New{+}T1$. In this case, Mss_n sends message *dereg* and waits for either the arrival of *deregAck* or the occurrence of timeout (T1), after which it decides what to do. When *deregAck* arrives before $New{+}T_1$ then through *h_RECD* Mss_n will learn if Mss_o has already processed the reply. If this is the case (solid lines) then at $New{+}T_1$ Mss_n will reply to Mss_{ini} without considering Mh_{new} (case a).[2] However, if Mss_o did not already process the reply for this M (dashed lines), then Mss_n will reset timer T1, re-forward M to Mh_{new}, wait for its reply, and send it to Mss_{ini} (case b). In the case that *deregAck* arrives after $New{+}T_1$, only if Mss_o has not already sent the corresponding reply, then Mss_n relies with NOk_M (case c).

5 Simulations

We have also prototyped and simulated AM^2C using the MobiCS[8] simulation environment. The purpose of the implementation was to get a deeper understanding of the complexity of the protocol and to evaluate its performance for different mobility scenarios. In this section we give an overview of the simulation environment, the AM^2C implementation and analyze some simulations results that we obtained.

[2]Assuming that Mss_n has no other $Mh \in$ *local_Mhs* $\cap$ *M.Dest*, then the reply will be Ok_M

5.1 MobiCS

MobiCS [8] is a Java framework for prototyping and simulating distributed protocols in mobile networks. The framework provides means for modular programming of distributed protocols and with interchangeable simulation models which do not affect the implementation of the distributed protocols.

MobiCS 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, where each one 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, reconnections, or in the specific case of AM^2C, 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 MobiCS 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 a well-defined functionality 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 exchanges through the wireless interface;
- **Hand-off** handles all (wired or wireless) messages related to a host migration.

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 AM^2C in MobiCS. We first declared each protocol message (e.g. `Dereg`, `DeregAck`, `Ack` etc.) and its attributes, then defined the corresponding handlers at the appropriate micro-protocols (e.g. `whenDereg`, `whenDeregAck` and `whenAck`) and finally declared the classes implementing these handlers. For example, we declared a class `AM2CMss` which implemented all the *message-handling* methods of the interfaces `AM2CMssWired`, `AM2CHandoffModule` and `AM2CMssWireless`.

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 external events (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 functions of its 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 that is sufficient for all the scenarios. Then one designs each scenario, 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.

We simulated AM^2C for several critical scenarios of concurrent migrations and multicasts. For example, we tested if the protocol executed correctly the cases (a), (b) and (c) mentioned in section 4.

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 situations tested in the deterministic mode.

5.3 Stochastic Simulations

With the stochastic simulations we wanted to evaluate the performance of AM^2C when used in networks with different mobility ratios. In particular, we measured (a) the percentage of aborted multicasts, (b) the number of additional *Ack* messages generated in Phase II, (c) the mean number of wireless messages per multicast and *Mh* and (d) the mean duration of a multicast. We measured each of these variables for 5 migration probabilities $P_{mig} = \{0.01, 0.2, 0.4, 0.6, 0.8\}$, and for two different values $\{125, 175\}$ for timeouts T1/T2, which were measured in *Simulated Time Units - STUs.*

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

- The static part of the network was composed of either 4 or 8 *Mss*s, (i.e. 4 or 8 cells), fully interconnected;
- The mobile part of the network consisted of 15 mobile hosts, all with the same migration probability P_{mig}, and same disconnection and reconnection probabilities ($P_{disc} = 0.01$ and $P_{reconn} = 0.3$)[3]
- The wireless transmission latency was set 50 times the wired transmission latency, which has a default value of 1 STU;
- Multicasts (to all 15 *Mh*s) were requested with probability $P_{req} = 0.1$, every 100 STUs;

Percentage of Aborts

We first measured the percentage of multicasts that are aborted (figure 4).

The results confirmed our expectation that the percentage of Aborts grows with the increase of the migration probability, and showed that the percentage of Aborts grows more at the lower spectrum of migration probabilities flattening for higher probabilities. In fact, one can see that already for $P_{mig} = 0.2$, approximately 50% of the multicasts are aborted. This is a direct consequence of AM^2C's "pessimistic" approach, where multicasts are aborted whenever a migration during a multicast.

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

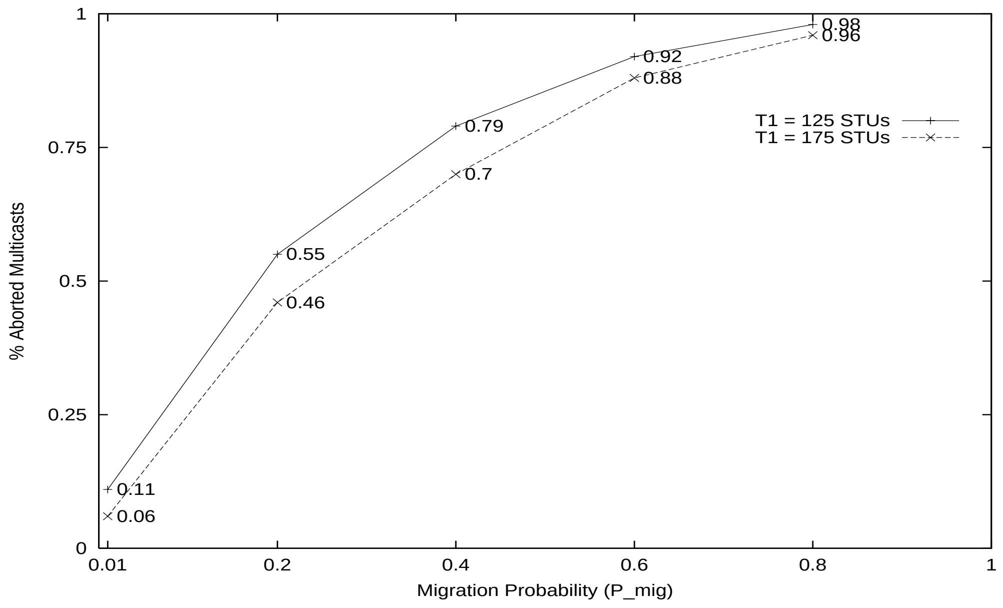

Figure 4: Percentage of Aborts

The curves at figure 4 also show that there are fewer Aborts for a larger timeout T1, as expected, but that this difference shrinks as the probability gets higher.

Additional Acknowledgements

In this test we evaluated how the number of additional *Ack* messages (generated in Phase II) relate to the migration probability, since these messages are the only variable part of the overhead in the wired network.

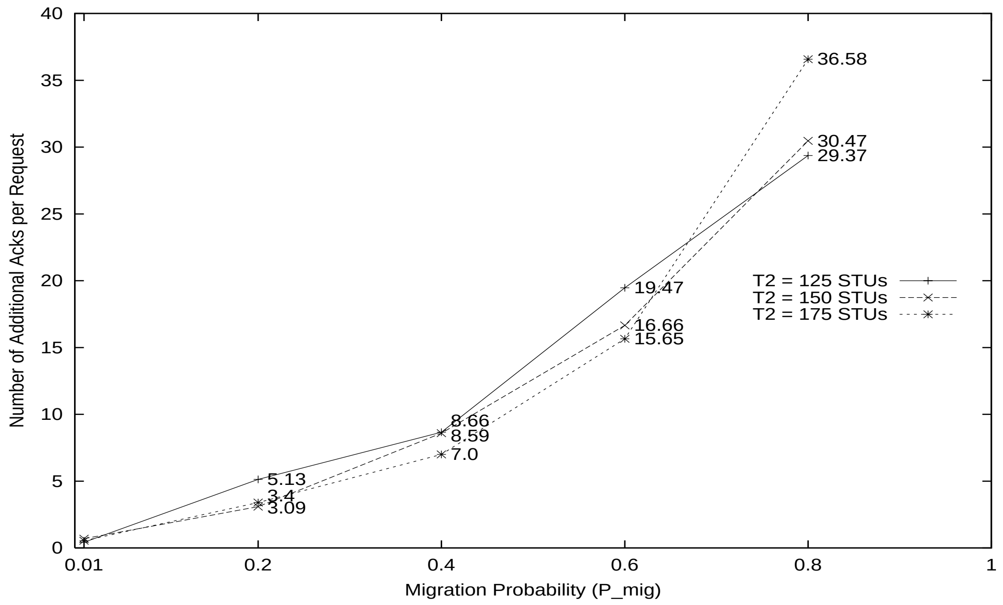

Figure 5: Number of Additional Acks

The graphics in Figure 5 shows that for small migration probabilities ($P_{mig} < 0.4$), the additional number of *Acks* remains reasonable (recall that all 15 *Mh*s are in M.Dest), but gets prohibitive for large values of P_{mig}. It also confirms that for "reasonable" values of P_{mig} a larger value of timeout T2 tends to reduce the number of *Acks*.

Wireless Messages

We also measured the mean number of wireless messages (per request and per *Mh*). As with the wired messages, we wanted also to check how much the number of wireless messages increases with the migration probability, and how much it depends on the timeout period.

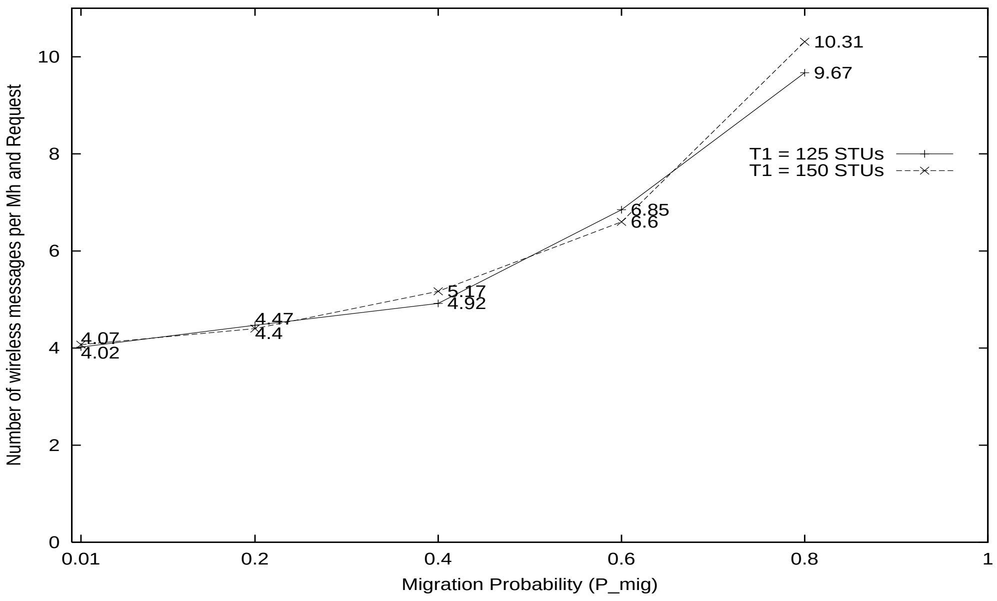

Figure 6: Number of Wireless Messages

The graphics of Figure 6 showed a rather slow increase of the curve, where the mean wireless messages only increases significantly beyond 4 after migration probability > 0.5. These results suggest that AM^2C does not create high overhead on the wireless medium. The graphics also shows that the value of timeouts T1 (and T2) only makes some difference for high migration probabilities.

Duration of the Multicast

Finally, we measured also how much the timeout (T1/T2) values and the migration probability have influence on the duration of the protocol. The duration was measured in STU, from the moment Mss_{ini} broadcasts a new multicast until the Del_M message is broadcasted to all the *Mss*s.

As can be seen from the graphics (Figure 7) the smallest timeout value (125 STU) caused the shortest protocol duration for any migration probability, despite the fact that it probably causes more *Mh*s to miss AM^2C's phase I or II messages (which in turn extends the protocol execution due to retransmissions) during ongoing migrations. The graphics also shows that AM^2C has a relatively small execution time, considering that each multicast requires at least 200 STUs just for the wireless transmissions (two transmissions in each phase, each lasting 50 STUs).

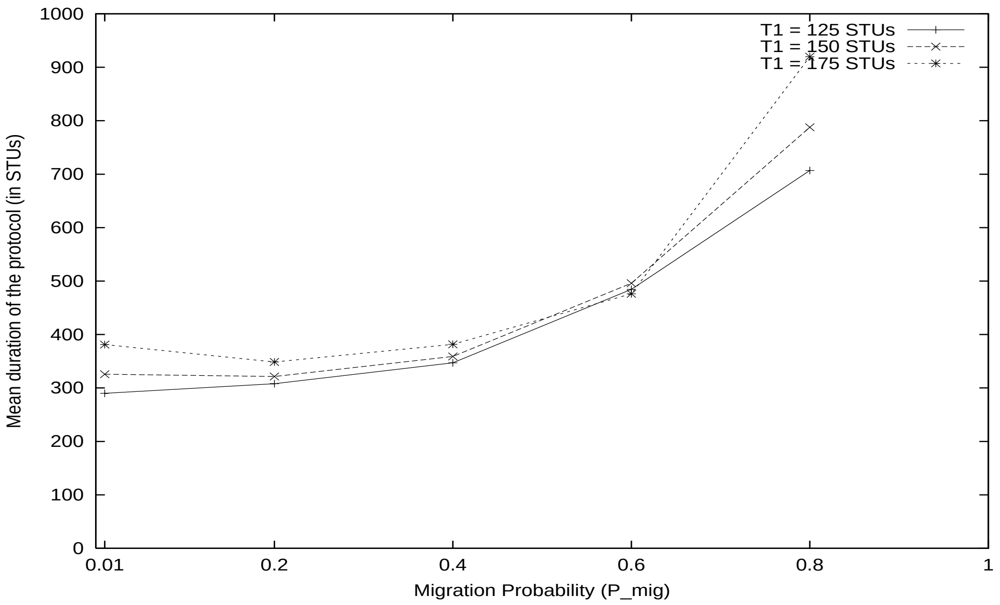

Figure 7: Duration of a Multicast

6 Related Work

We are unaware of any other work which deals with *atomic multicast* for mobile hosts, i.e. which propose a multicast service implementing *all-or-nothing* delivery semantics. Most other works propose protocols which only guarantee *reliable* multicast, i.e. that messages are eventually delivered to all the destinations (*Mh*s), but where the addressee do not get a feedback if 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 *Mss*s as intermediate repositories for pending multicasts. The main difference is that in our protocol the multicast message *per se* is deleted after the First phase (after timeout $T1$), and only the final status of the multicast (*outcome*) is retained until it has been delivered to all addressee.

Prakash et al. present an efficient causal ordering algorithm[14], 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 carry information about its direct predecessor messages with respect to each destination hosts. Hence, in a similar way their algorithm could also be incorporated into AM^2C, ensuring causally ordered, atomic 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 [15] and differ in the message complexity of the Hand-Off component and the size of the message headers (i.e. which 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 Mhs.* Among the proposed algorithms, only the first one is similar with MCAST (and with 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 to include sequence number information in the beacon messages, and *Mh*s to explicitly request re-transmission of "missed" messages through *Negative Acks (NACK)* messages. This not only waists 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 works [17, 11, 10] extend 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* multicasting and suffer from the scalability problems of Mobile IP. Mobile IP currently defines two basic approaches to support multicast: *remote subscription* and *bi-directional tunneling.*

Work by Harrison *et al* [11] proposes an approach called MoM for a Mobile IP-based Multicast service that addresses the problems of tunnel convergence and multicast duplication, by choosing a Designated Multicast Service Provider (DMSP) among all the home agents that have mobile host in a foreign network. In [16] several alternatives in the choice of the DMSP are considered and their performance was evaluated using a discrete event simulator. All these approaches, however, do not deal with loss of multicast packages during migrations or periods of inactivity of *Mh*s.

Maffeis *et al.* [12] describe a configurable, generic multicast service at the transport layer, which offers order-preserving best-effort multicast for different communication protocols. It is implemented as a set of so called *GTS servers*, which communicate with other GTS servers or the clients and guarantee message delivery in total order. Total order is achieved by defining a single GTS server as the sequencer for every multicast group. Messages sent to temporarily unavailable mobile hosts are kept in a message spool at the sequencer until they can be delivered to the destination. Thus, during disconnection of a group member all messages following the first non-delivered message are retained in the spool for future delivery. Our protocol differs from their approach in that it guarantees atomic delivery, but no total ordering policy. Other differences are that AM^2C does not rely on a single sequencer/spooler and is able to proceed with other multicasts even if some mobile hosts are unavailable.

7 Conclusion

In this paper we have presented an atomic multicast protocol for mobile hosts, which we called AM^2C. It is a 2PC protocol which guarantees that multicast messages are only be delivered to all the destinations (mobile hosts) if all of them are available and reachable during the first phase of the protocol, otherwise, the multicast is aborted. However, the multicast outcome ($Comm_M$, $Abor_M$) is transmitted to all group members until all of them acknowledge its receipt.

Compared to an earlier presentation of the protocol[9], the current version of the protocol has been refined, implemented and thoroughly tested. The preliminary simulation results showed that AM^2C's Abort ratio is quite sensitive the migration ratio of the mobile hosts, which suggests that the protocol is only suitable for networks high request-to-migration rates. On the other hand, it has a relatively short protocol execution time, which favors time-constrained group communication. It also causes a small overhead on the wireless links, and except for the wired messages of the hand-off protocol, also does not cause much overhead on the wired links.

One should say also that our simulation results are based on "theoretical" parameters (e.g. P_{mig}, P_{req}), which may not be realistic values. Hence, our simulations cannot be used to say if AM^2C would be feasible in a real setting, e.g. a wireless LAN. However, although we did not simulate the protocol for larger network configurations, it should be obvious that AM^2C is not scalable in the number of *Mss*s. Therefore, we are now working on a variant of the protocol, with a hierarchical *Mh* location management architecture, which we believe, will be more scalable than AM^2C.

As a future step we plan to implement the multicast service in a real wireless LAN environment for PDAs, investigate possible optimizations of AM^2C and design other multipoint communication protocols with weaker consistency properties.

Acknowledgments

We would like to thank Ricardo C.A. da Rocha for his patience in answering questions related to MobiCS and the very efficient support that he gave to the users of the MobiCS environment. We also thank Fabio Kon for his valuable comments concerning this work.

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 Intl. 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):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 and Distributed Systems*, 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):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. To appear.

[7] K. P. Birman. *Buiding Secure and Reliable Network Applications*. Manning Publications Company, May 1997.

[8] Ricardo C.A. da Rocha and Markus Endler. MobiCS: An Environment for Prototyping and Simulating Distributed Protocols for Mobile Networks. In *Proc. 3rd IEEE Intern. Conference on Mobile and Wireless Communications Networks (MWCN2001), Recife - Brazil*, pages 44–51, August 2001.

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

[10] 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.

[11] 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 Inter. Conference on Mobile Computing and Networking (Mobicom 97), Budapest, Hungary*, pages 151–160, September 1997.

[12] S. Maffeis, W. Bischofberger, and K. Mätzel. A generic multicast transport service to support disconnected operation. In *2nd USENIX Symposium on Mobile and Location-Independent Computing*, 1995.

[13] C.E. Perkins (editor). IP Mobility Support. RFC 202, IBM, October 1996.

[14] 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.

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

[16] C.L. Williamson, T.G. Harrison, W.L Mackrell, and R.B. Bunt. Performance evaluation of the mom mobile multicast protocol. *ACM Baltzer Journal on Mobile Networks and Applications*, 3(2):189–201, August 1998.

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