To appear in the 13th Intl. Conf. on Distributed Computing Systems, May '93

Delivering Multicast Messages in Networks with Mobile Hosts

Arup Acharya B. R. Badrinath
Department of Computer Science
Rutgers University
{acharya@riches, badri@cs}.rutgers.edu

Abstract

There is a strong trend towards integrating portable computers within existing data networks. Traditional network protocols were designed assuming a static view of network connectivity. A ***mobile host*** *can connect to the network from different locations at different times. This has led to the emergence of a new set of problems with regard to addressing schemes and network protocols for accommodating mobility within existing networks. This paper presents a protocol for delivering a multicast message* ***exactly once*** *to a group of mobile destinations. The protocol is based on a system model derived from an architecture developed in [5] for mobile internetworking, relying on "mobile support stations" within the fixed network to communicate with mobile hosts.*

1. Introduction

There has been considerable interest recently to extend data networks to handle mobility with the aim of providing continuous network connectivity to mobile hosts in spite of a change in their location. To handle mobile hosts, the fixed network is augmented with *mobile support stations/base stations* that act as access points to the fixed network for the mobile hosts. Research on the necessary software support required to handle mobility has focussed on addressing schemes and routing [5, 2, 7, 9], distributed file systems for mobile clients [8, 6] and handling database queries in mobile distributed environments [4]. So far, the work on addressing schemes and routing for mobile hosts has concentrated on how to assign addresses to mobile hosts so that IP-based protocols may be used for routing messages to a *single* mobile host. Using an abstract model of the architecture developed in [5] for mobile internetworking, this paper presents a protocol for efficiently *multicasting* a message to multiple mobile destinations. The M_CAST protocol presented here, ensures that a multicast message will be delivered to a mobile destination *exactly once* even though it may move to different mobile support stations during the execution of the protocol. Applications of this protocol include delivery of a mail message to a group of mobile users according to a distribution list associated with the mail.

Briefly, the architecture presented in [5] for mobile internetworking is as follows. The term "mobile" implies *able to move while retaining its network connections* [5]. A host that can move while retaining its network connections is a *mobile host* (MH). The infrastructure machines that communicate directly with the mobile hosts are called *mobile support stations*(MSS). A*cell* is a logical or geographical coverage area under a MSS. All MHs that have identified themselves with a particular MSS as belonging to its cell, are considered to be *local* to the MSS. A MH can *directly* communicate with a MSS (and vice versa) only if the MH is physically located within the cell serviced by the MSS. To communicate a datagram to another MH that is not in the same cell, the source MH contacts its local MSS which forwards the datagram to the local MSS of the target MH over the wired network. The receiving MSS then transmits the message over the wireless network to the target MH. At any given instant of time, a MH may belong to only one cell. The MSS *beacon protocol* is executed whenever a MH moves to a new cell. Addressing schemes for mobile hosts and routing protocols for (unreliable) delivery of *point-to-point* messages are presented in [5].

The system model that we use in this paper (Fig. 1), is derived from the architecture of [5] described above. This model consists of two distinct sets of entities: a large number of mobile hosts and a set of fixed hosts, some of which act as mobile support stations. Each MSS is capable of directly communicating with mobile hosts located within its cell via a wireless medium. All fixed hosts and the communication paths between them constitute the *static/fixed* network. Each MSS and the local MHs within its cell form a "wireless" network. Thus, based on the architecture of [5], our model consists of a static network along with a wireless network attached to each MSS that allows direct communication between a MSS and the MHs located within its cell. However, our system model diverges from the architecture of [5] by assuming that the fixed network guarantees reliable,

[1] This work was supported in part by the NSF under grant IRI-90101174 and is a component of the DATAMAN project at Rutgers, which addresses issues of data management in wireless mobile computing.

sequenced delivery of messages between any two MSSs, with a finite but unbounded message latency; similarly, the wireless network within the local cell of a MSS, ensures fifo delivery of messages from the MSS to a local MH. When a MH moves from one cell to another, a *hand-off* procedure is executed by the MSSs of the two cells. A MH is associated with a fixed address, its *id*, regardless of its location in the network. Lastly, all static and mobile hosts and communication links are assumed to be reliable.

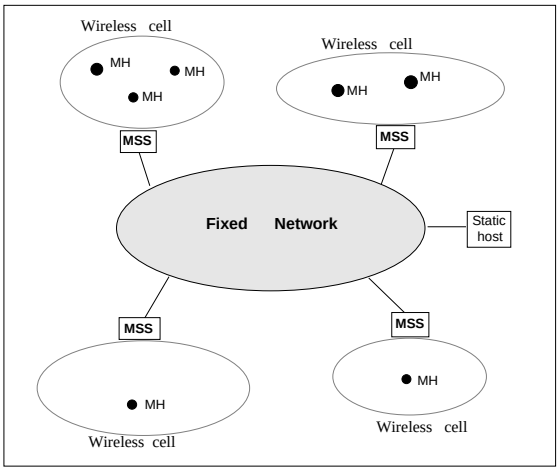

Fig. 1

Using the above system model for incorporating mobile hosts within a static network, we first identify the problems in delivering a multicast message exactly once, that arise due to the mobility of its recipients. We then present M_CAST, a protocol that delivers a multicast message *exactly once* to a specified set of destination MHs.

2. Motivation

We now consider some of the issues that are unique to a mobile computing environment. Due to limitations of memory, computing power and battery life at a portable computer, it is desirable that the computation required at a MH be kept to a minimum in any protocol execution. The MH should only be required to send and receive datagrams in sequence from its local MSS. A portable computer may not have a hard disk (e.g. palmtops, laptops) and since it may be powered off at arbitrary times by the user (especially, when not in use to lower power consumption), the necessary state information for a MH needs to be maintained at the local MSS and not at the MH itself. Further, a MSS should be required to maintain the state information of only those MHs that are currently located within its cell; it need not keep track of the location (and state information) of any non-local MH.

Besides these inherent constraints of mobile computing, the problem of delivering a multicast message *exactly once* to mobile destinations stems from two factors: first, the same MH may connect to the fixed network from different MSSs at different times, and secondly, copies of the same message sent over the wired network from a common (fixed) source may reach its destination MSSs at different times due to network latency. As an illustrative

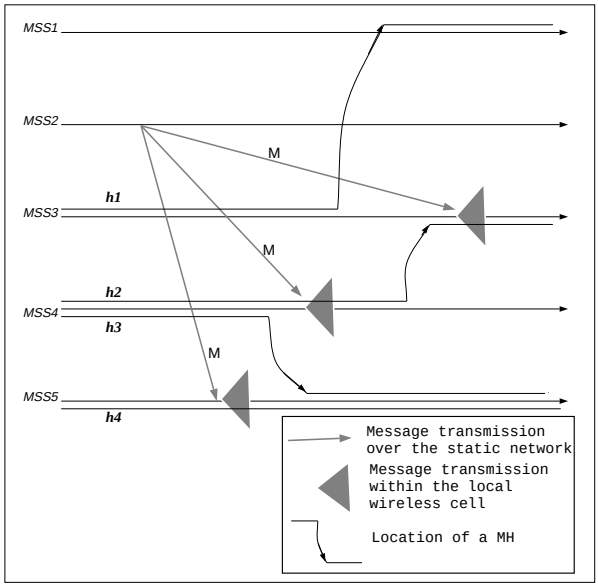

Fig. 2

example (Fig. 2), consider the delivery of a multicast message *M* sent from MSS2. The intended recipients of *M* are the MHs *h1*, *h2*, *h3* and *h4*. Initially, *h1* is local to MSS3, *h4* is local to MSS5, and MHs *h2* and *h3* are local to MSS4. Assuming that this information is available at MSS2, it forwards a copy of *M* (along with a list of intended recipients, *Dests(M)*) over the *fixed network* to just those MSSs that handle at least one of the target MHs, i.e. MSS3, MSS4 and MSS5. Each recipient MSS is now responsible for forwarding *M* over its wireless cell to those local MHs that belong to *Dests(M)*.

– Due to network latency, *M* is delivered to MSS4 after *h3* moved away to MSS5's cell. However, by the time *h3* moved into MSS5's cell, MSS5 had already received *M* and had delivered *M* to its local MHs, viz. *h4*. Thus, due to a change in its location, a mobile host such as *h3* may totally miss receiving a copy of *M*, even though the respective MSSs at its current and previous cells receive a copy *M*.
– Consider the MH *h2*. When *M* reached MSS4, *h2* was local to its cell and was forwarded a copy of *M*. Then, *h2* moved into the cell under MSS3, which did not receive *M* over the fixed network by that time. On receipt of *M*, MSS3 will forward a copy to *h2*, since *h2* is a local MH that is included in *Dests(M)*. Thus, *M* may be delivered more than once to *h2*.
– When *M* reaches MSS5, it is delivered to *h4* over the local wireless network. At this time, in spite of the

fact that there are no intended recipients of *M* within the local cell (that haven't received *M*), *M* cannot be deleted from the local buffer at MSS5. This is because an intended recipient of *M*, such as *h3*, may move into the local cell at a later time without having received a copy of *M* at a previous MSS. This also begs the question as to how long a multicast message needs to be buffered at a MSS. Thus, a recipient MSS needs to be explicitly informed by the multicast protocol to delete a message such as *M* from its local buffer, after it has been ensured that every MH in *Dests(M)* has received a copy of *M*.

– MSS2 did not send a copy of *M* to MSS1 since, according to the information at MSS2, no intended recipient of *M* was local to MSS1. However, *h1* moved to the cell under MSS1 before *M* reached MSS3. This points to the need for sending a copy of *M* to *all* MSSs; since the mobility of a MH can not be predicted a priori, it might move to any cell.

Here, we assumed that the sender of the multicast message, MSS2 has up-to-date information regarding the current location of each intended recipient of *M*, while transmitting the message to prospective MSSs. It is not always possible to maintain up-to-date information at *every* MSS regarding the location of *each* MH. Further, as the example suggests, the location of the intended destinations may have changed by the time the message reaches the individual MSSs. Thus, a sender MSS has two options: (a) send a copy of the message *M* to all MSSs, which will then forward it to appropriate local MHs or, (b) use a *point-to-point* message delivery protocol to individually deliver copies to each mobile destination. The M_CAST protocol presented here, is based on option (a). Before presenting M_CAST, we next consider the second option, which essentially simulates multicasting by unicasting copies of the same message to every destination.

First, note that since a copy of the message will be sent to many MSSs, the same physical link between two fixed hosts may be traversed many times, each time by a different copy of the same message. The work of propagating the same message to multiple (fixed) destinations over the static network, can be reduced by arranging the destinations into a spanning tree; this requires that the message be addressed to the set of destinations as a whole, instead of executing a message delivery protocol for each destination. Additionally, in a system with mobile hosts, simulating a multicast by multiple unicasts, one for each intended recipient, may well incur a much higher message overhead than for networks with only static hosts. To see why, consider the mechanism presented in [5] for (unreliable) delivery of a message *M* from a MH *h* to another MH *k* that is not in the same cell, but within the same *campus*. *M* is first transmitted to the local MSS, MSS_h, which then assumes responsibility for delivering *M* to *k*. If MSS_h knows the id of the MSS, viz. MSS_k, currently handling the MH *k*, then it suffices to send *M* over the fixed network to MSS_k which can then deliver *M* to *k* over the local wireless network. However, consider the following situations:

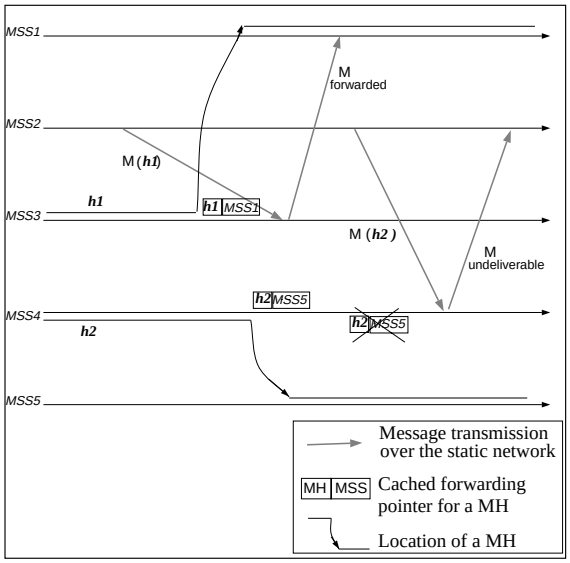

Fig. 3

– If *k* moves to another cell under MSS_l, then MSS_k maintains a forwarding pointer to MSS_l in its *cache*. In this case, on receiving *M*, MSS_k forwards *M* to MSS_l. It is possible that for a highly mobile host, *M* may be forwarded through multiple pointers. In fig. 3, this is illustrated by *h1* moving from the local cell of MSS3 to that of MSS1. When MSS3 receives *M* addressed to *h1*, it forwards *M* to MSS1, since the forwarding pointer for *h1* (pointing to MSS1) is still present MSS3's cache.

– When *M* is received at MSS_k, if the forwarding pointer to MSS_l (for messages intended for *k*) is no longer in its cache, then *M* is sent back to MSS_h. In fig.3, this situation occurs when *M* addressed to *h2* is received at MSS4. Since the forwarding pointer for *h2* is no longer present in the cache, *M* is returned back to MSS3 as undeliverable.

As is evident from above, *M* is further redirected from the recipient MSS in both cases. A much heavier penalty is incurred in terms of message overhead, when MSS_h does not have a forwarding pointer for the target MH *k* in its cache, or when *M* is returned back as undeliverable, as from MSS_k above. Now, MSS_h has to contact every MSS (only within the same *campus*) to find out the current location of *k*. It is also conceivable that this information is outdated by the time it reaches MSS_h especially if the network latency is considerable (e.g. if

the MH has moved to a new location in the meantime). This could lead to another round of multicast (over the fixed network) by MSS_h, and so on. Finally note that this message overhead is incurred just to deliver a message to a *single* destination; the overhead multiplies when the message delivery protocol is repeatedly executed, once for each destination.

3. The M_CAST protocol

3.1 Overview

The M_CAST protocol is initiated from a MSS to deliver a message *M* to *Dests(M)*, a set of destination MHs, with the guarantee that the message be delivered to each destination exactly once. The MSS initiating an execution of the protocol to deliver *M* will be referred to as the *initiator*, or MSS_{M_init}. Each MSS maintains its own local counter to assign sequence numbers to multicast messages initiated by it, which is incremented whenever it initiates a new execution of M_CAST. *M_id* refers to the message-id of *M*; it consists of the initiating MSS's id, *M_init*, concatenated with the sequence number, *M_seq*, assigned to *M* by its initiator. A MH may also invoke an execution of M_CAST: in this case, the local MSS initiates the protocol on behalf of the MH.

An execution of M_CAST consists of three distinct interactions, each handled by a different module. The WIRED module handles the initial exchange of control messages over the fixed network between the *initiator* and the *participant* MSSs. The WIRELESS module schedules transmission of the multicast messages over the wireless cell to its local MHs and informs the initiator (over the fixed network) of the list of local MHs to which a multicast message has been delivered. When a MH moves from one cell to another, the HAND-OFF module transfers state information associated with the MH, from the MSS of the previous cell to that of the new cell.

All messages exchanged between MSSs over the fixed network are assumed to be delivered in sequence with a finite, but arbitrary delay. Each MSS is equipped with a wireless interface to communicate with mobile hosts within its local cell; this interface is responsible for delivering messages in sequence to a local MH using protocols appropriate for the specific wireless communication medium in use. All three M_CAST modules use the wired network interface of a MSS to exchange messages over the static network; additionally, the WIRELESS module relies on the wireless interface to send messages to local MHs, and receive notification from the wireless interface on delivery of such a message to a local MH.

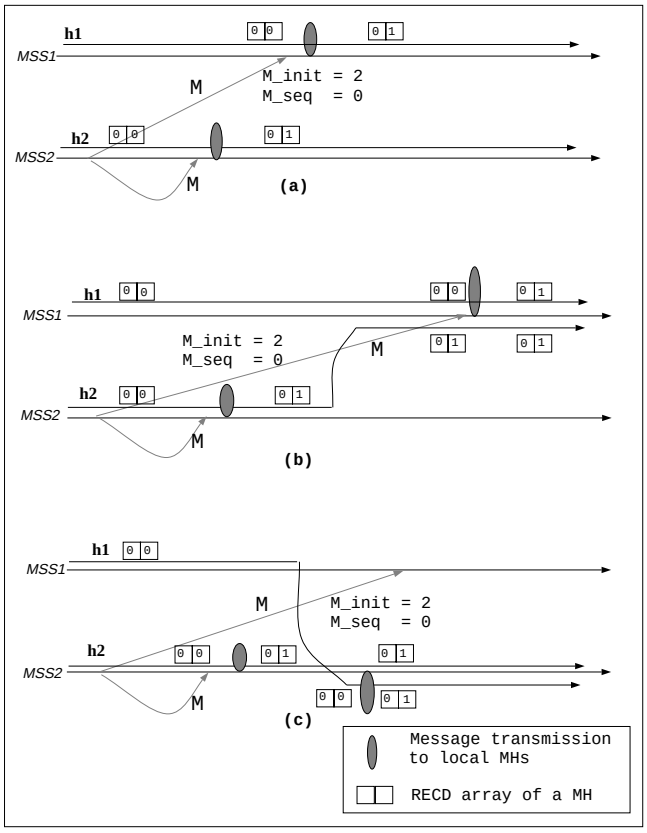

Fig. 4

The basic idea behind M_CAST is simple. The protocol associates an array *h*_RECD[1...N] with every MH *h*, which is kept *only* with the local MSS. This array records the sequence number of the next multicast message from each of the *N* MSSs in the system, that might include *h* as a recipient. The HAND-OFF module hands over this array to the MSS of the new cell when a MH switches its cell. When a message *M* is received at the local MSS from MSS_i and its sequence number, *M_seq*, is equal to *h*_RECD[i], then it is delivered to *h* over the local wireless cell and *h*_RECD[i] incremented if *h* is listed in *Dests(M)*. If *h* is not included in *Dests(M)*, then *h*_RECD[i] is incremented without delivering *M* to *h*. In fig.4, M_CAST is initiated from MSS2 to deliver *M* to the MHs *h1* and *h2*. Let the sequence number assigned to *M* by MSS2 be 0. In (a), *M* is forwarded to *h1* by MSS1 on receipt of *M*, since *M_seq* equals *h1*_RECD[M_init]. If *h1* was not included in *Dests(M)*, then MSS1 would simply increment *h1*_RECD[2] on receipt of *M*. In (b), *h2* moves to the local cell of MSS1 having already received *M* from MSS2. Since *h2*_RECD[2] is now greater than *M_seq*, MSS2 does not deliver *M* to *h2* for a second time. In (c), *h1* moves into the local cell of MSS2 with *h*_RECD[2] equal to 0; thus, MSS2 delivers *M* to *h1*.

M_CAST is a two-phase protocol. In the first phase,

the initiating MSS sends a copy of a message *M* and *Dests(M)*, a list of destination MHs, to all MSSs over the fixed network. Each recipient MSS is responsible for forwarding *M* over its wireless network to the local MHs that belong to *Dests(M)*. A recipient MSS then sends a *reply* message to the initiating MSS containing an ack for every local MH to which it could deliver a copy of *M*. Since a destination MH may already have received a copy of *M* in an earlier cell, a recipient MSS must deliver *M* to only those local MHs listed in *Dests(M)* that have not received it in a previous location. On the other hand, a MH listed in *Dests(M)* might move into a new cell, without having received *M* in any previous cell. However, in the new cell, the MSS may already have delivered *M* to its local MHs. Thus, a MSS needs to keeps *M* "live" (even though it my have delivered it to its local MHs) till it is notified by the initiator to do so otherwise. When the initiator receives an ack for every destination MH, it informs all MSSs to delete *M* from their respective message buffers, i.e. *M* is no longer "live" since it is no longer possible that a MH will move into a new cell without having received a copy of *M*.

To multicast a message from a fixed host to a group of destinations, which include both fixed and mobile hosts, a copy of the message is first sent via the fixed network to each static destination. A copy of the message along with a list of mobile destinations is then sent to some MSS; this MSS invokes M_CAST to deliver the message exactly once to each mobile destination. To multicast a message from a MH, the message is first forwarded to the local MSS. This MSS sends a copy of the message to each static destinations via the fixed network, and then invokes M_CAST to deliver the message to mobile destinations.

3.2 Data structures

At MSS_i, the list of MHs that are local to its cell is maintained as $Local_i$. An array *h*_RECD[1. . .N] with each element initially set to zero, is maintained for every local MH *h*. The significance of the value *h*_RECD[j] is that any message *M* multicast from MSS_j with a sequence number less than *h*_RECD[j] has been delivered to *h* if *Dests(M)* included *h*. The text of a multicast message *M* together with a list of local MH ids, $ack\_list_i(M)$, that have acknowledged receipt of *M*, is kept in $buffer_i$; this is later deleted on receipt of *delete(M)* message from MSS_{M_init}. A queue of multicast message-ids and their target local MHs are kept in $transmit_i$; the wireless interface is responsible for transmitting these messages over the local cell to the corresponding MHs, and notify the WIRELESS module on successful delivery of a message to a local MH.

MSS_i keeps track of the sequence number of the most recently received multicast message from each MSS in an array, mss_recd_i[1. . .N], with each element initially set to zero. The significance of the value mss_recd_i[j] is that the sequence number of the next multicast message expected from MSS_j is mss_recd_i[j]; all multicast messages with a lower sequence number have been received (over the fixed network) at MSS_i.

The data structures are shared by all the three modules of M_CAST. To ensure consistency of the data structures, it will be assumed that a module will lock all shared data structures, perform relevant updates and then release the lock.

3.3 WIRED module

To initiate an execution of M_CAST for a message *M*, the WIRED module at MSS_{M_init} executes the following actions:

1. Send a message over the wired network *multicast(text of M, Dests(M), M_id)* to all MSSs (including itself).
2. Create an empty list, *ack_recd(M)*. For each $ack\_list_i(M)$ received from MSS_i, the ids of the MHs listed in $ack\_list_i(M)$ are added to *ack_recd(M)*. When *ack_recd(M)* equals *Dests(M)*, a *delete(M)* message is sent to all MSSs.

On receipt of *multicast(M)*, the WIRED module of a recipient MSS, say MSS_i, executes as follows :

1. Increment mss_recd_i[*M_init*].
2. *M* is inserted in $buffer_i$ along with *Dests(M)*. A list, $ack\_list_i(M)$, initially empty, is also associated with *M* to keep track of the local MHs that have acknowledged receipt of *M*.
3. Create a list *local_dests(M)*, initially empty.
   a. $\forall h \in Local_i$, **if** (*h*_RECD[*M_init*] = *M_seq*) **then**
      if ($h \in Dests(M)$) **then** insert *h* in *local_dests(M)* **else** increment *h*_RECD[*M_init*]
      else
      if (*h*_RECD[*M_init*] > *M_seq*) **and** ($h \in Dests(M)$) **then** delete *h* from *Dests(M)*
   b. If *local_dests(M)* is not empty, then insert <*M_id*, *local_dests(M)*> in $transmit_i$, i.e. *M* is enqueued for transmission[2] over the wireless network to all local MHs listed in *local_dests(M)*. The $transmit_i$ queue is kept sorted in increasing order of message sequence numbers, with ties broken in favour of the one with a higher initiator-MSS id. Thus, *M_id*

[2] The wireless medium is physically a broadcast medium and therefore, a single message transmission should be targeted at as many local destinations as possible.

is used to correctly insert <*M_id, local_dests(M)*> in $transmit_i$.

4. Upon receipt of the *delete(M)* message at MSS_i, the entry *(text of M, Dests(M), M_id, $ack\_list_i(M)$)* is deleted from $buffer_i$.

Wireless interface : The wireless interface is not strictly a part of M_CAST. However, its role needs to be defined in the overall context of the protocol. First, it looks up the entry, say <*M_id, local_dests(M)*>, at the head of $transmit_i$ and transmits *M* over the wireless network to *local_dests(M)*. The wireless medium is capable of supporting broadcast transmissions; however, M_CAST does not specify whether *M* is sent in the local wireless network via point-to-point messages to each destination MH or as a broadcast transmission to all local MHs out of which only those listed in *local_dests(M)* pick up the message and respond with an ack. On receipt of such an ack, the wireless interface notifies the WIRELESS module.

The wireless interface is thus responsible for delivering messages to MHs in sequence using protocols suitable to the specific wireless medium in use.

3.4 WIRELESS module

The wireless interface notifies the module of a successful delivery of a message *M* to a MH *h* within the local cell of MSS_i, resulting in the following updates :

1. Delete *h* from *local_dests(M)* in $transmit_i$; if *local_dests(M)* now becomes empty, then the entry <*M_id, local_dests(M)*> is removed from $transmit_i$.
   Insert *h* in $ack\_list_i(M)$;
   Increment *h*_RECD[*M_init*].
2. If the condition

   $Dests(M) \bigcap Local_i \subseteq ack\_list_i(M)$ **and** $ack\_list_i(M) \neq \emptyset$

   is satisfied, then at the present time, there are no local MHs that belong to *Dests(M)* and have not yet received a copy of *M*. Further, there is atleast one local MH whose ack has not been sent back to the initiator. A message containing $ack\_list_i(M)$ is now sent to MSS_{M_init} and the lists, *Dests(M)* and $ack\_list_i(M)$, are re-initialized as :

   a. $Dests(M) := Dests(M) - ack\_list_i(M)$
      i.e. the copy of *Dests(M)* stored in $buffer_i$ no longer contains those local MHs to which *M* has been delivered.
   b. $ack\_list_i(M) := \emptyset$

   At a participant, MSS_i, the updates to *Dests(M)* by the WIRED and WIRELESS modules are performed on a copy of *Dests(M)* kept in $buffer_i$. Since the initiator MSS is also a participant (it receives a copy of *multicast()*), the WIRED module at MSS_{M_init} must perform both as the initiator and a participant; the WIRED module in the initiator mode uses a private (read-only) copy of *Dests(M)*.
3. **while** ($mss_recd_i[M_init] > h$_RECD[*M_init*])
   { Let *M'* denote the message from MSS_{M_init} with sequence number *h*_RECD[*M_init*].
   if *M'* $\in$ $buffer_i$ **and** *h* $\in$ *Dests(M')*
   then if $transmit_i$ contains an entry <*M'_id, local_dests(M')*>, add *h* to *local_dests(M')*; else such an entry is inserted in $transmit_i$ with *local_dests(M')* initialised to *h*.
   Exit *while* loop.
   else *h*_RECD[*M_init*] is incremented. }

3.5 HAND-OFF module

This module relies on the *beacon protocol* of [5] to detect that a new MH has moved into a cell. Each MSS periodically broadcasts its identity (*beacon*) in the local cell. When a MH receives a beacon packet different from the previous packet it received, it responds with a *greeting* message to the new MSS containing its own identity and the identity of the MSS of the previous cell. The MSS then acknowledges receipt of the *greeting* message to the new MH.

Using the beacon protocol, the HAND-OFF module first detects the entry of a new MH *h* in the local cell of MSS_i. Then, MSS_i sends a *deregister(h)* message to MSS_j, the MSS previously handling *h*. On delivery of this message, the HAND-OFF module at MSS_j executes as follows:

1. The WIRELESS module may have delivered messages to *h* before it left the cell, but for which the corresponding updates to the *h*_RECD array (at MSS_j) hadn't been completed. The HAND-OFF module waits till the updates to the *h*_RECD array (corresponding to the messages already delivered to *h*) are completed. Note that since no new messages can be delivered to *h* in MSS_j's cell, the number of pending updates to *h*_RECD array are bounded.
2. If any entry in $transmit_j$, say for message *M'*, contains *h* in its *local_dests(M')* list, then *h* is deleted from the list; if such an entry is at the head of $transmit_j$, then the wireless interface is also informed of *h*'s departure from the local cell given that it is no longer possible to deliver any message to *h* in this cell. The entire entry is removed if *local_dests(M')* becomes empty due to *h*'s deletion.

3. Send a message *register(h, h*_RECD*)* to MSS_i; then delete the array *h*_RECD[] and remove *h* from $Local_i$.

The HAND-OFF module at MSS_i takes the following actions on delivery of *register(h, h*_RECD*)* :

1. $\forall M' \in buffer_i$
   if $h \in Dests(M')$ **and** $M'\_seq <$ *h*_RECD[M'_init]
   then delete *h* from *Dests(M')*
2. **for** k = 1 to N **do**
   while (*h*_RECD[k] < mss_recd_i[k]) **do**
   if $buffer_i$ contains a message *M* such that
   M_init = k **and** M_seq = *h*_RECD[k] **and**
   $h \in Dests(M)$
   then
      if $transmit_i$ contains an entry <M_id, $local\_dests(M)$>
      then *h* is inserted in $local\_dests(M)$
      else <M_id, $local\_dests(M)$> is inserted in $transmit_i$, with $local\_dests(M)$ initialised to contain *h*.
   else increment *h*_RECD[k]
3. Insert *h* in $Local_i$, and install the *h*_RECD array received with the *register()* message, in the system structures.

4. Correctness

Assumptions : It is possible that a MH that changes its cell so frequently that it does not reside within a cell long enough to receive any (multicast) message transmissions over the local wireless network. We introduce the following realistic assumption to eliminate such an occurrence :

If for a given mh *h*, every MSS has at least one "deliverable" message *M* in its *buffer* (not necessarily the same message), i.e.

- *h*_RECD[M_init] = M_seq and
- $h \in Dests(M)$

then, *h* may make only a finite number of moves without receiving any "deliverable" message.

Multicast messages are transmitted over the wireless cell in the order in which they are inserted in the *transmit* list. It is assumed that these messages are received in the same sequence at a target MH. Before a MH moves away from its current cell, it is required that it acknowledge receipt (to the local MSS) of all those multicast messages that it received within this cell.

Correctness sketch : First, note that a message *M* will eventually be received by every MSS and will be kept in their respective buffers till an explicit *delete(M)* message is received. This *delete()* is sent by the initiator MSS only after it has received an ack for the delivery of *M* to every MH listed in *Dests(M)*.

Second, *M* is delivered to a MH *h* only if *h* belongs to *Dests(M)* and *h*_RECD[M_init] = M_seq, i.e. it is "deliverable". On delivery of *M* to *h*, the local MSS increments *h*_RECD[M_init]. Consequently, even if *h* were to move away to a new cell after acknowledging receipt of *M*, the *h*_RECD[] array will be handed over to the MSS of the new cell only after *h*_RECD[M_init] has been updated, i.e. the updated *h*_RECD[] array received at the new cell as part of handoff, makes it known to the new MSS that *h* has already received *M*. Thus, the role of *h*_RECD[] is to ensure *atmost once* delivery of a multicast message to *h*.

Finally, consider how could it be that a message *M* is never be delivered to a target mh *h*. Assume that a copy of *M* is present at every MSS and is "deliverable". Earlier, it was stated that a mh may change its cell only a finite number of times without picking up *any* "deliverable" message. Then, without violating this assumption, the only way through which *M* never gets to be delivered to *h* is if it were to always pick up deliverable messages other than *M*, i.e. *M* is "starved out"[3]. To prevent such an occurrence, the HAND-OFF module always schedules the transmission of messages "deliverable" to *h* in increasing order of messages sequence numbers (either by adding *h* to $local\_dests(M)$ in an existing entry for *M* in the *transmit* queue, or by inserting a new entry for *M*). To facilitate this process, the WIRED module also inserts entries to the *transmit* queue in increasing order of message-ids; this ensures that, at a later time, if MH *h* were to enter a cell and messages *M* and *M'* were both "deliverable" to *h* with $M\_seq < M'\_seq$, then the HAND-OFF module will not encounter a situation where *M'* is ahead of *M* in the *transmit* queue. Thus, M_CAST ensures that, once a message *M* has been received by every MSS, it will be delivered to a target MH before the delivery of any message (to the same MH) with a sequence number greater than M_seq. This ensures *atleast once* delivery of *M* while the use of *h*_RECD array guarantees *atmost once* delivery: together, M_CAST ensures *exactly once* delivery of multicast messages.

5. Performance issues

In a single execution of M_CAST, the initiator must send a *multicast()* message and later a *delete()* message to all MSSs; this requires *2N* messages over the fixed network. Additionally, a participant MSS replies to the initiator with *ack_list()* messages. For MSS_i to send a

[3] *Mobility* of recipients plays a very crucial role here, e.g. consider that *h* shuttles between the local cells of MSS1 and MSS2 such that it resides in any cell long enough to receive only one message transmission. Let MSS3 issue multicast messages at such a frequency that whenever *h* switches cells, a message from MSS3 is available for delivery to *h* at the new cell. Now, if "deliverable" messages are scheduled for transmission in increasing order of initiator-MSS ids, then delivery of messages with initiator-MSS ids greater than MSS3 could be forever postponed.

ack_list$_i$(M) message, the following condition needs to be first satisfied :
Dests(M) $\bigcap$ *Local$_i$* $\subseteq$ *ack_list$_i$(M)* **and** *ack_list$_i$(M)* $\neq \emptyset$
i.e., MSS_i will send a non-empty *ack_list$_i$(M)* message only if there is no local MH that is an intended recipient of *M* but has not yet received *M*. The role of this condition is therefore, to collect a maximal set of acknowledgments at MSS_i before forwarding them to the initiator. However, in the worst case, depending on when *M* reaches every participant MSSs and the mobility of the destination MHs, the number of *ack_list()* messages sent may equal |*Dests(M)*|.

In our system model, besides the messages sent over the fixed network, the number of wireless messages is also an important parameter of performance. The M_CAST protocol associates a list of local MHs, viz. *local_dests(M)*, with an entry for *M* in the *transmit* queue. This allows for a message *M* to be sent to multiple local MHs with a single transmission using the broadcast capability of the wireless medium. However, as with sending *ack_list()* messages, a combination of message latency over the fixed network and mobility pattern of the individual target MHs could force a separate wireless message to be transmitted for each target MH.

The M_CAST protocol requires the initiator to contact every MSS. Reliable communication between the initiator and other MSSs can be implemented on top of "best-effort" multicast routing protocols for static-host groups, such as [3]. From a scalability standpoint, M_CAST is intended for delivering multicast messages to mobile hosts within a single *campus* and needs to be suitably modified for *inter-campus* operation.

6. Conclusions

This paper first presented a system model for incorporating mobile hosts within a network of fixed hosts, derived from the architecture of [5]. It was shown that mobility of hosts coupled with network latency, introduce a new dimension to the problem of reliably delivering multicast messages without duplication, even in the absence of failures. Multicast routing algorithms such as [3] and different message delivery abstractions[1] have hitherto been limited to delivering messages to static hosts. It is our thesis that even with a reliable environment, mobility of hosts offers a new set of problems that need to be solved in order to incorporate mobile hosts within static networks.

In this paper, we focussed on how delivery of multicast messages is affected by a change in location of a mobile recipient. The ability of a mobile host to connect to the fixed network from different locations at different times combined with varying message latencies between static hosts (MSSs) in the fixed network, allows for the possibility that a mobile host may receive a multicast message at more than one location or may altogether miss receiving the message. This paper presented the M_CAST protocol for efficiently multicasting a message to multiple mobile destinations. It ensures that a multicast message will be delivered to a mobile destination *exactly once* even though it may move between different MSSs during the execution of the protocol. The protocol maintained an array, *h*_RECD[*N*], for each mobile host *h* that is always stored within the static portion of the network; when *h* swicthes its cell, *h*_RECD is transferred to the local MSS of the new cell as part of the *handoff* procedure. The size of this array is proportional to the number of MSSs in the system and independent of the number of MHs; MHs can be expected to vastly outnumber MSSs.

Acknowledgements

The authors would like to thank Ajay Bakre, Navin Budhiraja, Brian Davison, Vipul Gupta, Tomasz Imielinski, James Kistler, Partha Pal, Krithi Ramamritham, Tony DeSimone, Girish Welling, Wei Zhao and the anonymous referees for their constructive comments.

Bibliography

[1] K. Birman, A. Schiper, and P. Stephenson. Lightweight causal and atomic group multicast. *ACM Trans. Comput. Systems*, 9(3):272–314, 1991.

[2] D. Cohen, J. B. Postel, and R. Rom. Ip addressing and routing in a local wireless network. Manuscript, July 16, 1991.

[3] S. E. Deering and D. R. Cheriton. Multicast routing in datagram internetworks and extended lans. *ACM Trans. Computers*, May, 1990.

[4] T. Imielinski and B. R. Badrinath. Querying in highly mobile distributed environments. In *18th Intl. Conference on Very Large Databases*, pages 41–52, 1992.

[5] J. Ioannidis, D. Duchamp, and G. Q. Maguire. Ip-based protocols for mobile internetworking. In *Proc. of ACM SIGCOMM Symposium on Communication, Architectures and Protocols*, pages 235–245, September 1991.

[6] James Kistler and M. Satyanarayanan. Disconnected operation in the coda file system. *ACM Trans. on Computer Systems*, 10(1), Feb. 1992.

[7] Yakhov Rekhter and Charles Perkins. Optimal routing for mobile hosts using ip's loose source route option. In *Internet Draft*, October 1992.

[8] Carl D. Tait and Dan Duchamp. Service interface and replica management algorithm for mobile file system clients. In *Proc. First Intl. Conf. on Parallel and Distributed Information Systems*, 1991.

[9] Hiromi Wada, Takashi Yozawa, Tatsuya Ohnishi, and Yasunori Tanaka. Mobile computing environment based on internet packet forwarding. In *1992 Winter Usenix*, Jan. 1993.