A Multicast Primitive for Mobile Hosts*

Xinfeng Ye,
Department of Computer Science,
University of Auckland,
New Zealand.

John A. Keane,
Department of Computation,
UMIST,
Manchester, UK.

Abstract

Due to network latency and the mobility of the host, many existing group communication protocols are limited to a static environment. This paper presents a multicast primitive for delivering multicast messages to mobile hosts. The primitive has the total ordering property which guarantees the ordering of message delivery. The protocol also guarantees that the messages are delivered to the mobile hosts exactly once. Sequence numbers and message buffers are used to cope with message duplication and message loss.

1 Introduction

Much research has been carried out to extend data networks to provide continuous network connectivity to mobile hosts in spite of change in their locations. The problems involved in delivering multicast messages to mobile destinations are discussed in [1]. It is shown that network latency and the mobility of the hosts introduce some new problems for delivering multicast messages without duplication and loss. Due to these problems, the protocol in [3] is limited to delivering messages to static hosts. In [1] a protocol is proposed for delivering multicast messages to mobile destinations. The protocol guarantees that a multicast message is delivered to a mobile destination *exactly once*.

In this paper a primitive for delivering multicast messages to mobile hosts is introduced. In comparison to [1] the multicast primitive in this paper not only guarantees that a message is delivered to a host *exactly once*, it also has the *total ordering* property per group which guarantees that the hosts in a group receive their messages in the same order.

2 System Model

The system consists of a set of mobile hosts and a set of mobile support stations. A *mobile host* (MH) is a machine which can move while retaining its network connection. A *mobile support station* (MSS) is a static machine. A *cell* is an area serviced by an MSS. All MHs, that have identified themselves with a particular MSS as belonging to the cell serviced by the MSS, are considered to be *local* to the MSS. Cells can overlap with each other. An MH in overlapping cells can only treat one of the MSSs servicing the cells as its local MSS.

An MH and an MSS can directly communicate with each other if the MH is local to the MSS. Communication between the MHs and the MSSs is achieved via a wireless medium, and has the FIFO property. The MSSs in the system are connected through a fixed non-FIFO wired network. The MSSs communicate with each other through the network.

Each MH owned by an organisation is assigned a permanent IP address. An MSS of that organisation maintains a database recording the information concerning the organisation's MHs. This MSS is called the *home* MSS of the organisation's MHs; and the MHs are called the *native* MHs of the MSS. The system consists of MSSs and MHs owned by different organisations. An MH can move to any cell in the system and communicate with its local MSS. When an MH moves to another cell, the home MSS of the MH is informed of the new location of the MH.

To simplify the presentation, it is assumed that only the MHs generate multicast messages. Each MH belongs to some multicast groups. The home MSS of each MH records the information on the group membership of the MH. The MHs in a group can reside in different cells. When an MH needs to send a multicast message, the MH contacts its local MSS. The local MSS starts the execution of the primitive for delivering the multicast message to the destination MHs. The execution of the protocol involves the home MSSs and the local MSSs of the destination MHs.

The MHs only (a) send and receive messages from their local MSSs, and (b) record the identity of their local MSSs and the sequence number of the last received message. The MSSs maintain necessary state information for each of their local MHs, and execute the protocol for delivering multicast messages on behalf of their local MHs.

The MHs move among the cells. The MSSs use the *beacon protocol* of [2] to detect the MHs that move into their cells. In the beacon protocol the MSSs periodically broadcast their identities to the MHs in their cells. If an MH moves to a different cell, the MH will receive the identity of the MSS servicing the cell. However, if an MH is in overlapping cells, it will receive the identities of several MSSs. An MH, say h, in overlapping cells remains local to its original local MSS, say S, as long as h is in the cell serviced by S. Thus, when h receives an identity which is different from S's, h sends an *inquire* message to S

*This work is supported by Auckland University under grant A18/XXXXX/62090/F3414040, and by the UK EPSRC under grant GR/J48979.

0-7803-3280-6/96/$5.00 ©1996 IEEE

Authorized licensed use limited to: The University of Auckland. Downloaded on November 3, 2008 at 15:42 from IEEE Xplore. Restrictions apply.

to check whether h is still in the cell serviced by S. If h receives a reply from S, it means that h is still in the cell serviced by S. In this case, h ignores the beacon message. If h does not receive the reply from S within a predefined time period, h must have moved out of the cell serviced by S. Thus, h (a) sends a *greeting* message containing its own identity and the identity of its previous local MSS (i.e. S) to the MSS servicing the cell where the MH currently resides, and (b) records the received identity as the identity of its local MSS. From the greeting message sent by h, the MSS knows that h has joined the cell; the MSS also knows the identity of h's previous local MSS. The state information of h will be transferred from its previous local MSS to its current local MSS.

3 The Protocol

3.1 Problems and Solutions

First the problems in delivering multicast messages to mobile hosts are discussed, and the solutions to these problems are given.

Delivery Ordering: For some applications, the multicast protocol must provide guarantees regarding delivery order. The multicast primitive in this paper has the per group total ordering property. For example, assume that M_1 and M_2 are two messages initiated by two different MHs, and are delivered to the MHs in a group. Total ordering requires that M_1 and M_2 are received by all the MHs in the group in the same relative order.

To enforce total ordering, the principle of the multicast protocol of Ameoba [3] is used. In the protocol, each multicast message sent to a group is given a sequence number. The sequence number assigned to a message is unique in a group and increases monotonically. The MSSs deliver the multicast messages to the MHs in ascending sequence number. These guarantee that the MHs in a group receive the multicast messages in the same relative order. Thus, the total ordering is guaranteed.

To generate the sequence number, each group has a *sequencer*. The sequencer is an MSS and is responsible for generating the sequence number for the multicast messages of a group. When an MSS sends a multicast message on behalf of an MH, firstly, the MSS sends a request to the sequencer to obtain a sequence number for the message. When the sequence number is received, the message is sent to the home MSSs of the destination MHs. This is because the home MSSs keep the location information of the MHs. The home MSSs will forward the message to the local MSSs of the destination MHs. These local MSSs will deliver the messages to the destination MHs.

Message Duplication and Message Loss: Since the MSSs might send a multicast message to their local MHs at different times, if an MH moves to a different cell, then the MH might either receive a message more than once or miss the message. For example: assume that (a) a message M is to be delivered to a group to which an MH, say h, belongs, and (b) h moves from a cell serviced by MSS_1 to a cell serviced by MSS_2. If MSS_1 sends M to its local MHs before h leaves, and MSS_2 sends M after h arrives, then h receives M twice; thus, M is duplicated. On the other hand, if MSS_1 sends M after h leaves, and MSS_2 sends M before h arrives, then h will not receive M.

Two measures are used to prevent message duplication. One measure is to make the MSSs use a point-to-point protocol to deliver messages to their local MHs. In this protocol, instead of multicasting a message to the local destination MHs, each time an MSS sends the message to a single local MH. This can prevent a moving MH from receiving a message again in its new cell.

The other measure is to make each MH record the sequence number of the last received message. The recorded number is denoted as *rec_seq_num*. When an MH moves to a new cell, the MH includes *rec_seq_num* in the greeting message sent to its new local MSS. Since (a) messages are delivered in the sequence number order, and (b) *rec_seq_num* is the sequence number of the last message received by the MH, messages whose sequence numbers are not greater than *rec_seq_num* must have been received by the MH. Thus, from *rec_seq_num*, the MSS knows which messages have been received by the MH.

To prevent message duplication, before an MSS sends a message, say M, to an MH, the MSS compares the sequence number of the message (denoted as $M[seq\_num]$) with *rec_seq_num* of the MH. If "$M[seq\_num] \leq rec\_seq\_num$" holds, it means that M has been delivered to the MH. Thus, the MSS will not send M to the MH. As a result, message duplication is avoided.

To prevent message loss, the multicast messages are stored in the buffers of some MSSs. A message will not be deleted from the buffers until each of the destination MHs has received the message. To know that a message has been delivered to an MH, an MH must acknowledge the receipt of each message.

Delivery Protocol: When an MH moves to a new cell, the home MSS is informed of the new location of the MH. However, the location information might be out-of-date due to (a) the delay in passing the location information to the home MSS, and (b) the movement of the MHs. When a message is to be delivered to an MH, say h, h's home MSS sends the message to h's local MSS. If the location information is out-of-date, when the message arrives at the local MSS, h has already moved to another cell. In this case, the message will be forwarded to the MSS which is h's new local MSS. If h has moved across several cells, message forwarding might have to be carried out several times before the message is delivered to h.

In order to allow the forwarding to be carried out, each MSS keeps a forward pointer *f_ptr(h)* for each MH h which has moved away from its cell. *f_ptr(h)* records the identity of the MSS which services the cell that h has moved to. *f_ptr(h)* is updated when the MSS is notified of the leaving of h by the new local MSS of h.

If an MH keeps on moving amongst the cells and a forwarded message cannot reach the local MSS of the MH before the MH moves to another cell, the message will not be delivered to the MH. To solve the problem, when an MSS forwards a message, the MSS also stores the message in its buffer. Thus, if the MH moves back to the MSS later, the message

Authorized licensed use limited to: The University of Auckland. Downloaded on November 3, 2008 at 15:42 from IEEE Xplore. Restrictions apply.

in the buffer will be delivered to the MH.

An MH, say h, might visit the same cell more than once when a message, say M, is being forwarded to h. Since messages must be delivered in their sequence number order, M cannot be delivered to h if some messages whose sequence numbers are less than M's have not been delivered to h[1]. Thus, M might not be delivered to h even if the local MSS has a copy of M. As a result, when h moves to another cell, M has to be forwarded to the new local MSS of h. As described above, the MSSs store messages in their buffers. Thus, if the old local MSS of h has previously forwarded M to the new local MSS of h, the old local MSS does not need to send M to the new local MSS again. To determine whether a message needs to be forwarded, each MSS keeps a set, $out(M)_h$, for each message M being forwarded to h by the MSS. $out(M)_h$ records the identities of the MSSs to which the MSS has forwarded message M. Before M is forwarded to the local MSS of h, $out(M)_h$ is checked first. If the intended destination MSS is already in $out(M)_h$, there is no need to send M to the MSS again.

Buffer Management: As described earlier, to guarantee message delivery, messages are stored in the buffers of the MSSs. Since the buffer space of the MSSs is limited, after a message, say M, is delivered to its destination MHs, M should be deleted from the buffers. *out(M)* sets on the MSSs record where M has been forwarded (i.e. where M has been stored). Thus, by linking the MSSs in *out(M)*, a chain (or several chains) of MSSs can be formed. Each MSS in the chain(s) has a copy of M. The home MSS of the destination MH of M is the MSS which sends out M at the beginning. Thus, the home MSS is the first element of the chain(s). Hence, starting from the home MSS, if we travel along the chain(s), we can reach all the MSSs which have a copy of M. This principle is used in finding M and sending *discard* messages to the MSSs which have a copy of M.

To determine whether a message has been delivered to all its destination MHs, an MSS is required to send an acknowledgement to the sender of the message (i.e. the local MSS of the MH which generates the message) on behalf of the recipient after the MSS has delivered the message to its local MH. However, this creates a lot of network traffic in a multicast operation. To reduce the network traffic, instead of requiring each message be acknowledged by its recipients, only some of these messages need to be acknowledged. For the messages which are required to be acknowledged, when the senders of these messages receive the acknowledgements from all the recipients, they instruct the MSSs to delete the messages from the buffers of the MSSs. Since all messages are delivered to the MHs in ascending sequence numbers, if an MH acknowledges the receipt of a message, say M, then the MH must have received all the messages whose sequence numbers are less than the sequence number of M. Thus, when the sender of M receives acknowledgements from all the recipients of M, the recipients must have received all messages whose sequence numbers are less than M's. Hence, when the MSSs are instructed to discard M, the MSSs also discard messages whose sequence numbers are smaller than M's sequence number.

The sequencer is responsible for deciding which sender should wait for acknowledgements from the recipients of its message. The decision is based on the number of multicast messages which are currently in the buffer and the buffer space on the MSSs. When the available buffer space of the home MSSs is below a threshold, the sequencer requires a sender to be responsible for deleting its message from the buffers of the MSSs. The sequencer keeps a record on the number of the multicast messages which have been sent. The record is updated when the sequencer issues a sequence number to a multicast message. The sequencer also keeps the sequence number of the last message which is required to be deleted from the buffer. From these two numbers, the sequencer can determine the number of messages which reside in the buffers. The buffer size of the MSSs is predetermined, and this information can be collected by the sequencer before the computation starts.

After successfully delivering a message to a local MH, an MSS reports the delivery to the sender of the message. After the sender has received the acknowledgement from all the recipients, the sender sends *delete* messages to the home MSSs of the recipients. The home MSS sends *discard* messages to the MSSs in *out(M)* to instruct them to delete M from their buffers. On receipt of the *discard* message, an MSS deletes M from its buffer. If the MSS has forwarded M to other MSSs, it will send *discard* messages to the MSSs in its *out(M)*. Thus, the *discard* messages are propagated to all the MSSs which have stored M in their buffers. As a result, M will be deleted from the buffers of all the MSSs.

Due to the non-FIFO property of the wired network, a multicast message, say M, might arrive at an MSS after the message instructing the deletion of M. As a result, M might not be deleted from buffers. To prevent this problem, each MSS keeps a variable $delivered\_seq\_num_h$ for each MH h which has visited the cell serviced by the MSS. $delivered\_seq\_num_h$ records the sequence number of the last message which is known to have been received by h. $delivered\_seq\_num_h$ at an MSS is updated either (a) when h sends a greeting message to the MSS (i.e. when h moves into the MSS's cell), or (b) when h acknowledges the receipt of a message sent by the MSS (i.e. h is local to the MSS), or,(c) when a *discard* message is received. Thus, only the local MSS of h has an accurate $delivered\_seq\_num_h$ value; the $delivered\_seq\_num_h$ of other MSS need not be up-to-date. A *discard* message instructs the deletion of a message, say M. If $delivered\_seq\_num_h$ of the MSS is less than M's sequence number, $delivered\_seq\_num_h$ is set to M's sequence number. Since messages are delivered to h in their sequence number order, when a message to be delivered to h arrives, if the sequence number of the message is less than $delivered\_seq\_num_h$, the message is deleted. As a result, all messages will be deleted eventually.

3.2 The Rules

An MSS starts running the primitive when it receives a request from an MH to multicast a message, say X. The MSS is called the *initiator* of both the

[1]This is because the wired network does not have FIFO property. Thus, messages might not arrive at an MSS in their sequence number order.

Authorized licensed use limited to: The University of Auckland. Downloaded on November 3, 2008 at 15:42 from IEEE Xplore. Restrictions apply.

primitive and X. The running of the primitive consists of two or three phases depending on whether the initiator needs to receive acknowledgements from the recipients of X. In the first phase, the initiator sends a request to the sequencer to obtain a sequence number for X. The sequencer allocates a sequence number to X and sends the number to the initiator. The sequencer also informs the initiator whether it will be responsible for deleting X from the buffers of the MSSs. In the second phase, the initiator assigns the sequence number received from the sequencer to X and sends the message to the home MSSs of the recipients of X. The home MSSs send X to the local MSSs of the recipients of X. X is stored in buffer and is delivered to its destination MH after all the messages whose sequence numbers are less than X's sequence number have been delivered. If the initiator is responsible for deleting X from the buffers of the MSSs, an acknowledgement is sent to the initiator. The third phase is only needed if the initiator is responsible for deleting X from the buffers of the MSSs. When the initiator has received all the acknowledgements for X, the third phase starts. In this phase, the initiator instructs all the MSSs to delete X from their buffers.

First the rule used by the sequencers is described. Each sequencer keeps two variables, *seq_num* and *deleted_seq_num*. *seq_num* records the latest sequence number that has been allocated to the multicast messages. It is initialised to 0. *deleted_seq_num* is the sequence number of the last message required to be deleted from the buffers of the MSSs. The initial value of *deleted_seq_num* is 0.

Message *seq_req* is sent by the initiators to obtain sequence numbers from the sequencers. When a *seq_req* message is received, the sequencer increments *seq_num* to obtain the next available sequence number. Since *seq_num* is increased by one each time, the sequence numbers are a sequence of consecutive integers. Thus, the difference of *seq_num* and *deleted_seq_num* indicates the number of messages which have not been required to be deleted from the buffers of the MSSs. A predetermined value, *threshold*, is used by each sequencer to determine whether an initiator should be responsible for instructing the MSSs to discard its message. When "*seq_num* − *deleted_seq_num*" exceeds *threshold*, some messages have to be deleted from the buffers.

A *sequence number message* (*seq_num, discard_value*) is used to sent the sequence number back to the initiator. *seq_num* is the sequence number requested by the initiator. *discard_value* indicates whether the initiator is responsible for discarding its message from the buffers of the MSSs. If the sequencer decides that an initiator is responsible for deleting its message, *discard_value* is set to *yes*. Otherwise, *discard_value* is set to *no*. If *discard_value* is set to *yes*, *deleted_seq_num* is set to the sequence number of the message to be deleted.

R1. **when receive a** *seq_req* **message from** MSS_j:
  $seq\_num \leftarrow seq\_num + 1$
  if $seq\_num - deleted\_seq\_num > threshold$ **then**
    $discard\_value \leftarrow yes$
    $deleted\_seq\_num \leftarrow seq\_num$
  else
    $discard\_value \leftarrow no$
  endif
  send (*seq_num, discard_value*) **to** MSS_j

Now the rules used by the initiators of the multicast messages are described. First the initiator obtains a sequence number for a multicast message, say X, from the sequencer (*R2*). When the sequence number message (*seq_num, discard_value*) is received from the sequencer (*R3*), *seq_num*, *discard_value*, the identity of the initiator, the identifier of the group to which X is sent, and X are combined together to form a message, say M. Then, M is sent to the home MSSs of the destination MHs. If the initiator is responsible for deleting M after M's delivery, i.e. $discard\_value = yes$, the initiator sets up a variable, ack_num_M, to record the number of acknowledgements to be received from the recipients of multicast message. Then, it waits for the acknowledgements.

After M is delivered to an MH, if the initiator of M needs to be acknowledged, an acknowledgement *ack(M)* is sent to the initiator. ack_num_M is decremented when an *ack(M)* arrives (*R4*). When ack_num_M is equal to zero, it means that all the destination MHs have received M. Thus, M can be deleted from the buffers of the MSSs. As a result, the initiator sends *discard(M)* to instruct the home MSSs of the destination MH of M to delete M from their buffers (*R5*).

R2. **when need to multicast a message:**
  send *seq_req* **message to the sequencer**

R3. **when receive sequence number message** (*seq_num, discard_value*):
  combine the multicast message, the identifier of the group receiving the multicast message, *seq_num, discard_value* **and the identifier of the initiator to form a new message** M;
  multicast M **to the home MSSs of the destination MHs of** M;
  if $discard\_value = yes$ **then**
    $ack\_num_M \leftarrow$ **the number of** M**'s recipients**
  endif

R4. **when receive an** *ack(M)*:
  $ack\_num_M \leftarrow ack\_num_M - 1$

R5. **when** $ack\_num_M = 0$:
  multicast *discard(M)* **to all the home MSSs of the destination MHs of** M

Now the rules used by the home MSSs are discussed. In the following discussion for a message M, $M[msg]$, $M[seq\_num]$, $M[discard\_value]$, $M[gp]$ and $M[init]$ denotes the multicast message, the sequence number of the multicast message, the *discard value* associated with the multicast message, the group identifier to which the message is sent, and the identifier of the initiator of the multicast message respectively.

When a home MSS receives a multicast message (*R6*), it sends the message to the local MSS of the recipients of the messages. An MSS does not know which group a local MH belongs unless the MSS is the home MSS of the MH. Thus, when a home MSS sends a message to the local MSS of a native MH, the home MSS also indicates the identity of the MH which will receive the message. Hence, the message being sent out is a pair (M, h) where M is the multicast message, and h is the identity of the MH. As

Authorized licensed use limited to: The University of Auckland. Downloaded on November 3, 2008 at 15:42 from IEEE Xplore. Restrictions apply.

described in §3.1, the identity of the local MSS will be recorded in *out(M)* to indicate where the message has been sent. Later, when a *discard(M)* message is received from the initiator of M (*R7*) for each native MH, say h, which is in the multicast group of *M[gp]*, the MSS sends *discard(M, h)* to all the MSSs in $out(M)_h$ to delete M being stored on these MSSs. As a result, $out(M)_h$ is set to empty to indicate that *delete* messages have been sent out.

R6. when receive a multicast message M:
  for each native MH h such that h is the destination MH of M do
    send (M, h) to h's local MSS, say MSS_j
    $out(M)_h \leftarrow out(M)_h \cup \{MSS_j\}$
  end_for_each

R7. when receive *delete(M)*:
  for each h such that h is a member of group *M[gp]* do
    for each MSS_j in $out(M)_h$ do
      send *delete(M, h)* to MSS_j
    end_for_each
    $out(M)_h \leftarrow \emptyset$
  end_for_each

Now the rules used by the MSSs to handle the receipt and the delivery of a multicast message are introduced. As described in §3.1, each MSS keeps a variable $discard\_seq\_num_h$ for each MH, say h, which has visited the MSS. $discard\_seq\_num_h$ records the sequence number of the last message that is known to have been received by h. Due to the non-FIFO property of the network, a message, say (M, h), might arrive at an MSS after h has received the multicast message (i.e. "$M[seq\_num] \leq delivered\_seq\_num_h$"); thus, (M, h) is deleted on its arrival. If (M, h) arrives normally, (M, h) is stored in the buffer (if necessary) to guarantee message delivery as described in §3.1. If (a) the recipient MH has moved to another cell (i.e. $h \notin locals$) and (b) the message has not been previously forwarded to the new local MSS of the MH (i.e. $f\_ptr(h) \notin out(M)_h$), then the message is forwarded to the new local MSS of the MH and $out(M)_h$ is updated accordingly.

R8. when receive a multicast message *(M, h)*:
  if $M[seq\_num] \leq delivered\_seq\_num_h$ then
    discard (M, h);
  else /* M arrives normally */
    if (M, h) is not in buffer then
      store (M, h) in buffer
    endif
    if $(h \notin locals) \wedge (f\_ptr(h) \notin out(M))$ then
      send (M, h) to $MSS_{f\_ptr(h)}$
      $out(M)_h \leftarrow out(M)_h \cup \{f\_ptr(h)\}$
    endif
  endif

When a *delete* message *delete(M, h)* is received, $delivered\_seq\_num_h$ is updated (if necessary). As described in §3.1, messages are delivered in their sequence number order. Thus, the messages which are sent to the same multicast group as M and whose sequence numbers are less than $M[seq\_num]$ must have been delivered to the MHs before M. Hence, these messages can also be deleted when (M, h) is deleted. After deleting a message, say (M', h'), from its buffer, an MSS, say MSS_i, will send a *delete* message to each MSS where MSS_i has forwarded a copy of M' (i.e. the MSSs in $out(M')$). This allows the *delete* message to be propagated to all the MSSs where M' is stored.

R9. when receive $delete(M, h)$:
  if $M[seq\_num] > delivered\_seq\_num_h$ then
    $delivered\_seq\_num_h \leftarrow M[seq\_num]$
  endif
  for each (M', h') in buffer such that $(M'[gp] = M[gp]) \wedge (M'[seq\_num] \leq M[seq\_num])$ do
    send *delete(M', h')* to all the MSSs in $out(M')_{h'}$;
    discard (M', h') from buffer;
    $out(M')_{h'} \leftarrow \emptyset$;
  end_for_each

According to *R1*, the sequence numbers generated by the sequencer are a sequence of consecutive integers. Messages are delivered in ascending sequence number order, and $delivered\_seq\_num_h$ records the sequence number of the last message delivered to an MH, say h. Thus, the sequence number of the next message to be received by h must be $delivered\_seq\_num_h + 1$. When a message is ready to be delivered to an MH h, i.e. $M[seq\_num] = delivered\_seq\_num_h + 1$, the portion which contains the original multicast message, i.e. *M[msg]*, is sent to h. If an MH h acknowledges the receipt of a message M, the MSS increments $delivered\_seq\_num_h$. According to *R3*, if "$M[discard\_value] = yes$" holds, the initiator is waiting for acknowledgements (*R3*). Thus, an acknowledgement *ack(M)* is sent to the initiator after h receives the message.

R10. when $(h \in locals) \wedge$ (there exists a message (M, h) in buffer such that $M[seq\_num] = delivered\_seq\_num_h + 1$):
  send $M[msg]$ to h through wireless medium
  if h acknowledges the receipt of M then
    $delivered\_seq\_num_h \leftarrow delivered\_seq\_num_h + 1$
    if $M[discard\_value] = yes$ then
      send $ack(M)$ to $MSS_{M[init]}$
    endif
  endif

The rule used by the MHs in message delivery is described below. Since the sequence numbers of the messages are consecutive integers, after receiving a message, the MH increments *rec_seq_num* to record the sequence number of the received message and acknowledges the receipt of the message.

R11 when receive a message *msg* from h's local MSS MSS_j:
  $rec\_seq\_num_h \leftarrow rec\_seq\_num_h + 1$
  send acknowledgement to MSS_j

Now the rules describing the operations occurred when the MHs change cells are given. As described in §2, when an MH, say h, receives a beacon message, h sends an *inquire* message to its current local MSS to check whether h is still within the MSS's cell. If h is still within the cell of its current lo-

Authorized licensed use limited to: The University of Auckland. Downloaded on November 3, 2008 at 15:42 from IEEE Xplore. Restrictions apply.

cal MSS, the MSS will send a reply to h. If h does not receive reply from its current local MSS, h must have moved to another cell. In this case, h reports its arrival by sending a greeting message to its new local MSS. The greeting message is a pair, say $(MSS_i, rec\_seq\_num_h)$, where MSS_i is the identifier of h's old local MSS, and $rec\_seq\_num_h$ is the sequence number of the last message received by h. From $rec\_seq\_num_h$, the new local MSS knows which messages have been delivered to h. Since $delivered\_seq\_num_h$ should record the sequence number of the last message received by h, $delivered\_seq\_num_h$ is set to $rec\_seq\_num_h$. The new local MSS of h will send a *migration inform* message $(MSS_i, rec\_seq\_num_h, h)$, where MSS_i is the new local MSS of h, to MSS_j (i.e. h's old local MSS). The message informs MSS_j of the identity of h's new local MSS (i.e. MSS_i). $rec\_seq\_num_h$ is also included in the message. This is because h might have received a message from MSS_j but moved away from MSS_j before sending an acknowledgement. If an acknowledgement is not sent, MSS_j will not be aware of the receipt of the message. From $rec\_seq\_num_h$ in the *migration inform* message, MSS_j can determine whether the last message has been received by h.

R12. **when receive a greeting message** $(MSS_j, rec\_seq\_num_h)$ **from an MH** h:
$locals \leftarrow locals \cup \{h\}$;
$delivered\_seq\_num_h \leftarrow rec\_seq\_num_h$;
send migration inform message $(MSS_i, rec\_seq\_num_h, h)$ **to** MSS_j

When an MSS, say MSS_i, receives a *migration inform* message, say $(MSS_j, rec\_seq\_num_h, h)$, if MSS_i did not receive the acknowledgement for the last message sent to h, then, according to $rec\_seq\_num_h$, MSS_i can determine whether h has received the last message (if any). "$rec\_seq\_num_h > delivered\_seq\_num_h$" holds on MSS_i if the last message is received by h. This is because, (a) according to *R11*, h increments $req\_seq\_num_h$ when it receives a message, and (b) according to *R10*, $delivered\_seq\_num_h$ is only incremented when an acknowledgement is received. If the last message has been received by h and the initiator of the message is waiting for an acknowledgement for the receipt of the message, MSS_i sends an *ack* message to the initiator of the message, and sets $delivered\_seq\_num_h$ to the sequence number of the last message received by h. "$rec\_seq\_num_h > delivered\_seq\_num_h$" does not hold if the last message is not received by h. This is the case if h has moved to another cell before receiving the last message sent by MSS_i. In this case, the new local MSS of h would be responsible for delivering the message to h.

When a *migration inform* message is received, h is removed from *locals* to indicate that h has moved to another cell. MSS_i should also updates the value of *f_ptr(h)* to point to the new local MSS of h. If there are messages in the buffer which are due to be delivered to h, these messages should be forwarded to the new local MSS of h (i.e. MSS_j). However, a message, say M', only needs to be forwarded to MSS_j if MSS_i has not previously sent M' to MSS_j (i.e. $MSS_j \notin out(M')_h$).

R13. **when receive a migration inform message** $(MSS_j, rec\_seq\_num_h, h)$:
if $(rec\_seq\_num_h > delivered\_seq\_num_h)$
$\wedge (M[discard\_value] = yes$ **where** M **is the last message sent to** h**) then**
send $ack(M)$ **to** $MSS_{M[init]}$
$delivered\_seq\_num_h \leftarrow rec\_seq\_num_h$
endif
$locals \leftarrow locals - \{h\}$
$f\_ptr(h) \leftarrow MSS_j$
for each (M', h) **in the buffer such that** $rec\_seq\_num_h < M'[seq\_num]$ **do**
if $MSS_j \notin out(M')_h$ **then**
send (M', h) **to** MSS_j
$out(M')_h \leftarrow out(M')_h \cup \{MSS_j\}$
endif
end_for_each

4 Conclusions

As the scheme in [1], the protocol in this paper guarantees that multicast messages are delivered to their destinations exactly once. In contrast to [1], which does not guarantee the delivery order of the messages, the primitive here guarantees that messages sent to the same destination MHs are delivered to the MHs in the same relative order per group even if the MHs are located in different cells. The approach to enforcing the ordering is based on the principles of the multicast protocol of Amoeba in [3]. However, the scheme in [3] is limited to delivering messages to static hosts. As a result, sequence numbers and buffers are used in this paper to cope with the possible message duplication and loss when multicasting messages to MHs.

Unlike the scheme in [1], which assumes that the wired network has the FIFO property, the protocol here uses sequence number to cope with message duplication. This allows the protocol to be applied to systems with non-FIFO wired network. The correctness proof and the message complexity of the scheme have been discussed in [4].

References

[1] A. Acharya and B.R. Badrinath, Delivering Multicast Messages in Networks with Mobile Hosts, Proc. of the 13th International Conference on Distributed Computing Systems, (1993) 292-299.

[2] J. Ioannidis, D. Duchamp and G.Q. Maquire, Ip-based protocols for mobile internetworking, Proc. of ACM Symposium on Communication, Architectures and Protocols, (1991) 235-245.

[3] M.F. Kaashoek and A.S. Tanenbaum, Efficient Reliable Group Communication For Distributed Systems, Submitted for publication, 1995

[4] X. Ye, A Multicast Primitive for Mobile Hosts, Research Notes, Department of Computer Science, Auckland University, 1996

Authorized licensed use limited to: The University of Auckland. Downloaded on November 3, 2008 at 15:42 from IEEE Xplore. Restrictions apply.