A Lightweight Algorithm for Causal Message Ordering in Mobile Computing Systems

Chakarat Skawratananond, Neeraj Mittal, and Vijay K. Garg

TR-PDS-1998-11 November 1998

Parallel & Distributed Systems group
Department of Electrical & Computer Engineering
University of Texas at Austin
Austin, Texas 78712

A Lightweight Algorithm for Causal Message Ordering in Mobile Computing Systems

Chakarat Skawratananond, Neeraj Mittal, and Vijay K. Garg*
Parallel and Distributed Systems Laboratory
http://maple.ece.utexas.edu
The University of Texas at Austin,
Austin, Texas 78712.
{chakarat,neerajm,vijay}@pine.ece.utexas.edu

Abstract

With the popularity of portable computers and the improvements of wireless networking, there is a great deal of interest in developing applications for mobile computing systems. Causally ordered message delivery is required in several distributed applications particularly those that involve human interactions (such as teleconferencing and collaborative work). In this paper, we present an efficient protocol for causal ordering in mobile computing systems. This protocol requires minimal resources on mobile hosts and wireless links. Our message overhead in wired network is low. The proposed protocol is scalable and can easily handle host connections and disconnections. Our protocol, when compared to previous proposals, offers a low unnecessary delay, low message overhead and optimized handoff cost.

1 Introduction

The emergence of mobile computing devices, such as notebook computers and personal digital assistants with communication capabilities, has had a significant impact on distributed computing. These devices provide users the freedom to move anywhere under the service area while retaining network connection. However, mobile computing devices have limited resources compared to stationary machines. For example, mobile devices have small memory space, limited power supply, and less computing capability. Furthermore, the communication between mobile devices and wired network employs wireless channels which are susceptible to errors and distortions. Also, the cost of using these wireless channels is relatively expensive. Distributed algorithms that run on the system with mobile computing devices therefore require some modifications to compensate for these factors.

In this paper, we consider causal message ordering required in many distributed applications such as management of replicated data [8, 9], distributed monitoring [6], resource allocation [18], distributed shared memory [3], multimedia systems [2], and collaborative work [19]. The protocols to implement causal message ordering in systems with static hosts have been presented in [14, 9, 16, 18, 20, 21]. These protocols can be executed by every mobile host with all the relevant data structures being stored on the mobile hosts themselves. However, considering limited resources and bandwidth of wireless links available to mobile hosts, it is not appropriate to apply these protocols directly to mobile systems. As introduced in [5], the following four factors should be taken into account in designing protocols for mobile systems.

*supported in part by the NSF Grants ECS-9414780, CCR-9520540, a TRW faculty assistantship award, a General Motors Fellowship, and an IBM grant.

1. The amount of computation performed by a mobile host should be low.
2. The communication overhead in the wireless medium should be minimal.
3. Algorithms should be scalable with respect to the number of mobile hosts in the system.
4. Algorithms should be able to easily handle the effect of hosts connections and disconnections.

While ordering of messages in distributed systems with static hosts has received wide attention, there has been little work on causal message ordering in mobile computing systems. Alagar and Venkatesan [5] proposed three algorithms based on the algorithm by Raynal, Schiper and Toueg (RST) in [18]. The first algorithm ($AV1$) maintains causal ordering among all mobile hosts. The message overhead is proportional to the square of the number of mobile hosts (n_h). However, the data structures required in the algorithm are stored in mobile support stations to reduce load on mobile hosts and wireless links. In the second algorithm ($AV2$), causal ordering is exclusively maintained among mobile support stations. The message overhead reduces to the square of the number of mobile support stations (n_s). However, the procedure for handling host migration (*handoff*) is more complicated than the first algorithm. Since stronger ordering is imposed, messages may experience unnecessarily delay even though they do not violate causal ordering in the mobile hosts' view. Their third algorithm ($AV3$) is aimed at reducing this unnecessary delay by partitioning each physical mobile support station into k logical mobile support stations. As k increases, the degree of unnecessary delay decreases, but the message overhead and the cost of handling host migration increases. The message overhead in $AV2$ and $AV3$ does not depend on the number of participating mobile hosts, they are therefore suitable for dynamic mobile systems.

Yen, Huang, and Hwang (YHH) [22] proposed another algorithm based on [18]. The message overhead in their algorithm lies between that of $AV1$ and $AV2$. In particular, each mobile support station maintains a matrix of size $n_s \times n_h$; this matrix is attached to each message sent. The unnecessary delay in their algorithm is lower than $AV2$. Their handoff module is also more efficient than $AV2$. The message overhead in their algorithm depends upon the number of participating hosts in the system. As a result, their algorithm is not scalable and unsuited for dynamic mobile systems.

Prakash, Raynal, and Singhal (PSR) [17] presented an algorithm to implement causal message ordering in which each message carries information only about its direct predecessors with respect to each destination process. Message overhead in their algorithm is relatively low; however, in the worst case, it can be as large as $O(n_h^2)$. Furthermore, the structure of their message overhead depends on the number of participating processes. This makes their algorithm unsuitable for dynamic systems.

In this paper we propose a new protocol suited to mobile systems in which message overhead is comparable to those for static systems, and limited resources on mobile hosts are efficiently utilized. Our protocol is also suitable for systems where the number of participating hosts is varied dynamically. Moreover, the proposed protocol is scalable since our message overhead structure is independent of the number of hosts in the system. Our contribution can be summarized as follows: (1) With our protocol, we are able to decrease the unnecessary delivery delay while maintaining low message overhead. In the worst case, the message overhead in the wired network is $O(n_s^2 + n_h)$. (2) Our handoff module is more efficient than $AV2$ and $AV3$ because we do not require the messages exchanged among mobile support stations to be causally ordered. (3) We provide proof of correctness for both the static and the handoff modules. The correctness proof becomes important in the light of the fact that we discovered a bug in YHH. In particular, their protocol, as presented in [22], does not satisfy the liveness property. (4) Finally, we state and prove the condition implemented by our static module. We also present conditions implemented by $AV2$ and YHH (corrected) algorithms.

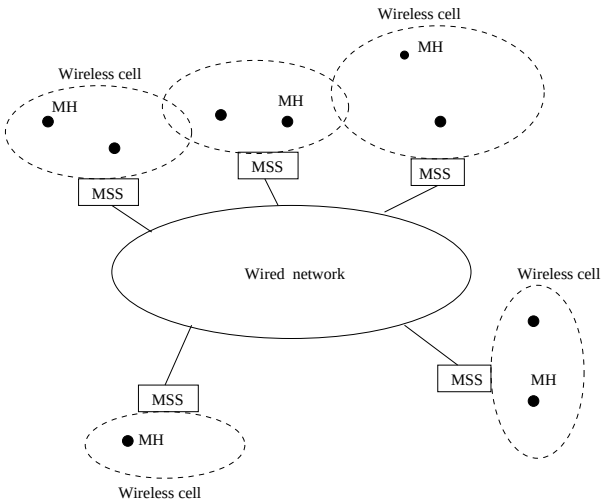


Figure 1: A mobile computing system.

The rest of the paper is organized as follows. Section 2 presents the system model and the notation used in the paper. Sufficient conditions for causal message ordering in mobile computing systems are presented in Section 3. We present our protocol in Section 4.1 (static module) and Section 4.2 (handoff module). The correctness proof is provided in Section 4.3. We compare our protocol with the previous work in Section 5. The simulation results are presented in Section 6. Section 7 concludes the paper.

2 System Model and Definitions

A mobile computing system consists of two kinds of processing units: *mobile hosts* and *mobile support stations*. A mobile host (MH) is a computer that can move while retaining its network connections. A mobile support station (MSS) is a machine that can communicate directly with mobile hosts over *wireless* channels. MSSs form the infrastructure of this system model. The geographical area which an MSS's wireless signal can cover is called a *cell*. Even though cells may physically overlap, we assume that an MH is directly connected through a wireless channel to at most one MSS at any given time. An MH can communicate with other MHs and MSSs only through the MSS to which it is directly connected. All MSSs and communication paths between them form the *wired* network. Figure 1 illustrates a mobile computing system. We assume that the wireless channels are FIFO, and both wired and wireless channels are reliable and take an arbitrary but finite amount of time to deliver messages. A mobile host can disconnect itself from the network and can reconnect at a later time.

Each *process* (MH or MSS) in a computation generates an execution trace, which is a finite sequence of local *states* and *events*. A state corresponds to the values of all the variables and the program counter in the process. An event on a process can be classified into three types: *send* event (corresponds to send of a message by a process) , *receive* event (corresponds to arrival of a message at a process), and *local* event (which is not a send or a receive event). A *delivery* event is a local event that represents the delivery of a received message to the application or applications running on that process.

Let $\mathcal{H} = \{h_1, h_2, \ldots, h_{n_h}\}$ represent the set of mobile hosts and $\mathcal{S} = \{S_1, S_2, \ldots, S_{n_s}\}$ denote the set of mobile support stations. In practice, $n_h \gg n_s$. Also, let $\mathcal{H}_i$ denote the set of MHs in the cell of MSS S_i. A mobile computation can be illustrated using a graphical representation referred to as *concrete diagram*. Figure 2 illustrates such a diagram where the horizontal lines represent MH and MSS processes, with time

progressing from left to right. h_1 is in the cell of S_1. h_2 and h_3 are in the cell of S_2. A solid arrow represents a message exchanged between an MH and an MSS process. A dashed arrow represents a message sent from an MSS process to another MSS process. Filled circles at the base and the head of an arrow represent send and receive events of that message. A concrete diagram in which only MH processes are shown is referred to as an *abstract diagram*.

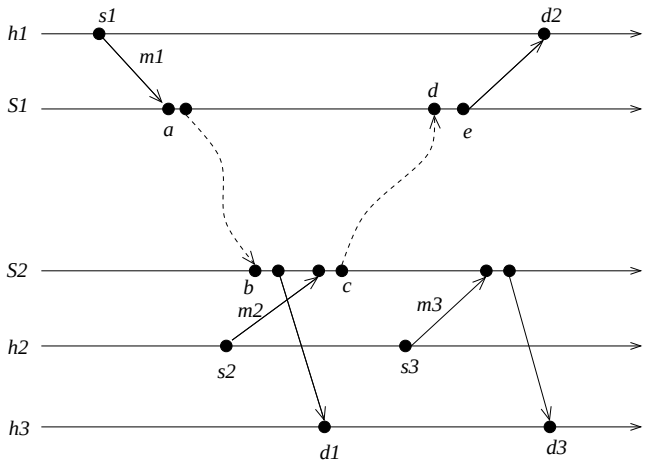


Figure 2: A concrete diagram of a mobile computation.

We denote the sequence of MSSs that an MH h_l visits by $\{S_k^l\}_{0 \leq k \leq n(h_l)}$, where $n(h_l)$ is the number of times h_l switches cell in a computation. Using this notation, S_0^l and $S_{n(h_l)}^l$ represent the *initial* and the *final* MSSs for h_l. Note that an MH can visit an MSS more than once. For a message m, let $m.src$ and $m.dst$ denote the source and destination processes. Moreover, $m.snd$, $m.rcv$ and $m.dlv$ denote the send event on the source process and the receive and the deliver events on the destination process respectively. We assume that a message sent to itself is immediately received by the sending process.

An *application* message is a message sent by an MH intended for another MH. Since MHs do not communicate with each other directly, an MH, say h_s, first sends an application message m to its MSS, say S_i, which then forwards m to the MSS, S_j, of the destination host, h_d. Using our notation, $m.src$ and $m.dst$ denote the source and the destination hosts respectively of m. In other words, $m.src = h_s$ and $m.dst = h_d$. Furthermore, $m.snd$ denotes the send event of m on h_s. Also, $m.rcv$ and $m.dlv$ denote the receive and delivery events respectively of m on h_d.

Let $\hat{m}$ denote the message which S_i sends to S_j (containing the application message m along with additional information for ensuring causality), requesting it to deliver m to h_d. Again using our notation, $\hat{m}.src$ denotes the MSS of h_s when m was sent (in this case S_i). Similarly, $\hat{m}.dst$ denotes the MSS to which S_i forwards m (in this case S_j). As before, $\hat{m}.snd$ denotes the send event of $\hat{m}$ on the support station S_i. Similarly, $\hat{m}.rcv$ and $\hat{m}.dlv$ (when m becomes deliverable at S_j) denote the receive and delivery events respectively of $\hat{m}$ on S_j. Figure 3 illustrates our notation.

An event e locally occurred before an event f in mobile host's view, denoted by $e \prec_h f$, iff e occurred before f in real-time on some mobile host. Similarly, an event e locally occurred before an event f in mobile support station's view, denoted by $e \prec_s f$, iff e occurred before f in real-time on some mobile support station. Let $\rightarrow_h$ and $\rightarrow_s$ denote the Lamport's happened before relation [15] in abstract (on events on MHs) and concrete diagram (on events on MSSs) respectively. A mobile computation is causally ordered iff the following property is satisfied for any pair of application messages, m_i and m_j, in the system,

$$(CO) \qquad m_i.snd \rightarrow_h m_j.snd \Rightarrow \neg(m_j.dlv \prec_h m_i.dlv)$$

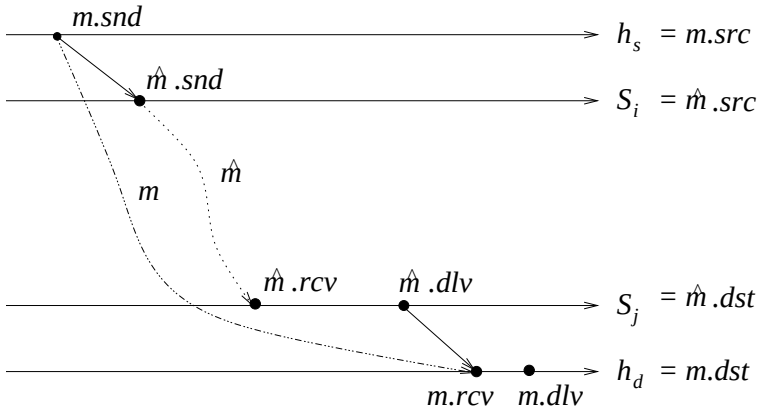


Figure 3: A figure illustrating the notation used in the paper.

For convenience, $m_i \rightarrow_h m_j \stackrel{def}{=} m_i.snd \rightarrow_h m_j.snd$.

3 Sufficient Conditions

We next give the sufficient conditions for causally ordered message delivery in a mobile computation with static hosts.

Theorem 1 *: A mobile computation is causally ordered if*

(C_1) all wireless channels are FIFO,

(C_2) messages in the wired network are causally ordered, and

(C_3) each MSS sends out messages in the order they are received.

Proof: The condition C_2 can be formally expressed as,

$$(CO') \qquad \hat{m}_i.snd \rightarrow_s \hat{m}_j.snd \Rightarrow \neg(\hat{m}_j.dlv \prec_s \hat{m}_i.dlv)$$

We first prove that $m_i.snd \rightarrow_h m_j.snd \Rightarrow \hat{m}_i.snd \rightarrow_s \hat{m}_j.snd$. Let $\rightsquigarrow_h$ and $\rightsquigarrow_s$ relate the send and delivery events of the same message in abstract and concrete views respectively. Observe that due to C_1 and C_3, $m_i.snd \prec_h m_j.snd \Rightarrow \hat{m}_i.snd \prec_s \hat{m}_j.snd$. Moreover, since MHs communicate through MSSs therefore $m_i.snd \rightsquigarrow_h m_i.dlv \Rightarrow \hat{m}_i.snd \rightsquigarrow_s \hat{m}_i.dlv$, and $m_i.dlv \prec_h m_j.snd \Rightarrow \hat{m}_i.dlv \prec_s \hat{m}_j.snd$. Using induction on the definition of $\rightarrow_h$, it can be easily proved that $m_i.snd \rightarrow_h m_j.snd \Rightarrow \hat{m}_i.snd \rightarrow_s \hat{m}_j.snd$ (any causal chain from $m_i.snd$ to $m_j.snd$ is a combination of the three cases). Informally, if there is a causal path from m_i to m_j in the abstract diagram then there is a causal path from $\hat{m}_i$ to $\hat{m}_j$ in the concrete diagram.

Again, due to C_1, we have $m_j.dlv \prec_h m_i.dlv \Rightarrow \hat{m}_j.dlv \prec_s \hat{m}_i.dlv$. Using contrapositive, we get $\neg(\hat{m}_j.dlv \prec_s \hat{m}_i.dlv) \Rightarrow \neg(m_j.dlv \prec_h m_i.dlv)$. Thus, $m_i.snd \rightarrow_h m_j.snd \Rightarrow \hat{m}_i.snd \rightarrow_s \hat{m}_j.snd \Rightarrow \neg(\hat{m}_j.dlv \prec_s \hat{m}_i.dlv) \Rightarrow \neg(m_j.dlv \prec_h m_i.dlv)$. In other words, assuming C_1 and C_3, $CO' \Rightarrow CO$. ■

Sufficient conditions given in Theorem 1 were implicitly used in [5]. For systems with static hosts, Theorem 1 gives a lightweight protocol for causal message ordering. In the extreme case when the entire

computation is in a single cell, causal ordering can be provided by simply using FIFO channels between MHs and their MSSs.

We now show that C_1, C_2, and C_3 are not necessary by a counter-example. In Figure 4, $s_1 \rightarrow_h s_3$ and $d_1 \prec_h d_3$. Therefore the computation in Figure 4 is causally ordered, although C_1 and C_2 do not hold.

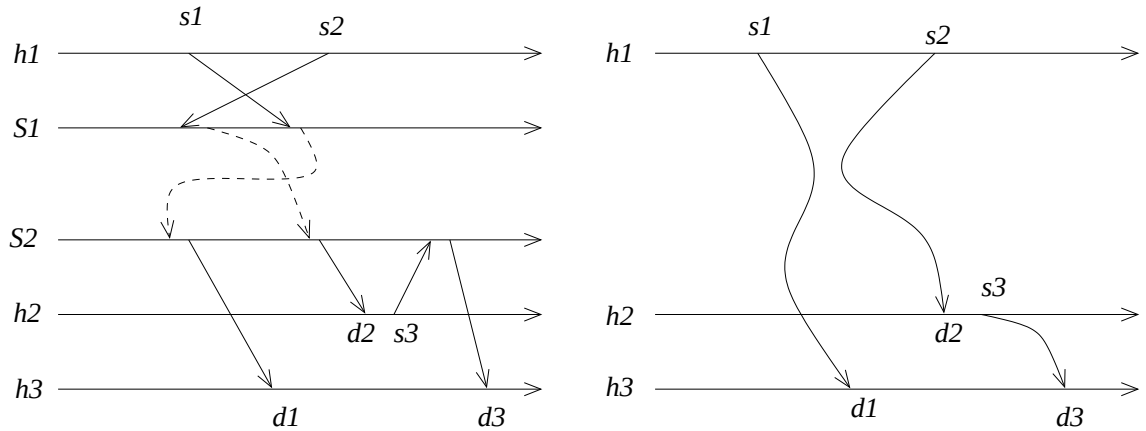


Figure 4: An example to show that C_1, C_2, and C_3 are not necessary for CO.

The algorithms presented by Alagar and Venkatesan ($AV2$ and $AV3$) [5] enforce CO' in order to achieve CO. Their algorithms delay messages that violate CO' even though they do not violate CO. This can be illustrated in a computation in Figure 5. In this example, message m_1 does not causally precede m_3 in the abstract view, but it does in the concrete view. Under CO', m_3 is unnecessarily delayed until m_1 is deliverable. Our goal is to reduce this unnecessary delay, while maintaining the message overhead in the wired network close to $O(n_s^2)$.

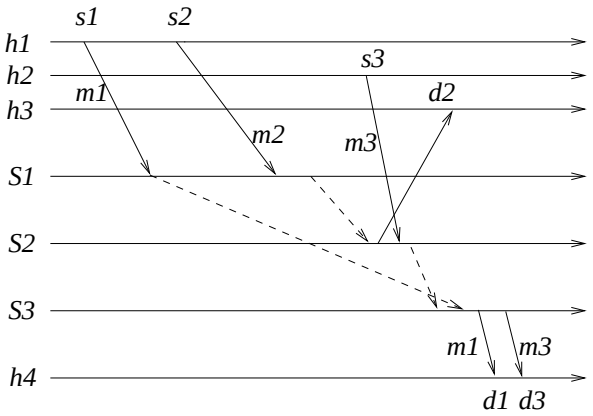


Figure 5: Unnecessary delay in $AV2$.

4 Algorithm

$AV2$ uses a single matrix for all MHs in a cell. This can create false causal dependencies between messages. In order to reduce these false causal dependencies and hence the unnecessary delay, we propose to use a

separate matrix for each MH in a cell. The next two subsections describe the static and the handoff modules of our protocol. The static module is executed when an MH is in a particular cell. The handoff module is executed when an MH moves from one cell to another. We prove the correctness of both the modules in Section 4.3. Section 4.4 presents the condition characterizing the static module.

4.1 Static Module

For convenience, we first describe the static module assuming static hosts. In the next subsection, we describe the handoff module and the modifications that need to be made to the static module to incorporate mobile hosts.

Our static module is based on the algorithm proposed by Raynal *et al* [18]. For simple exposition of the protocol, we assume that the channels among the MSSs are FIFO. This assumption can be easily relaxed by implementing FIFO among MSSs using sequence numbers. We also assume that every MSS knows about the location of the MHs. For each MH h_l, we maintain an $n_s \times n_s$ matrix M_l. $M_l[i,j]$ denotes the total number of messages h_l knows to have been sent by S_i to S_j. Assume that h_l is in the cell of S_i. In order to reduce the communication and computation overhead of h_l, the matrix M_l is stored at S_i. In addition, each S_i also maintains two arrays $lastsent_i$ and $lastrcvd_i$ of size n_s. The j^{th} entry of $lastsent_i$, $lastsent_i[j]$, denotes the number of messages sent by S_i to S_j. Similarly, the j^{th} entry of $lastrcvd_i$, $lastrcvd_i[j]$, denotes the number of messages sent by S_j that have been received at S_i.

Initially, all the entries in the matrices M_l, and arrays $lastsent_i$ and $lastrcvd_i$ are set to 0. To send a message m to another MH h_d, h_s first sends the message to its MSS S_i. Assume that h_d is in the cell of S_j. S_i increments $lastsent_i[j]$ by one and then sends $\langle m, M_s, lastsent_i[j]\rangle$, denoted by $\hat{m}$, to S_j. After that, S_i sets $M_s[i,j]$ to $lastsent_i[j]$.

S_j on receiving $\langle m, M, seqno\rangle$ from S_i meant for h_d first checks whether m is deliverable. m is *deliverable* if S_j has received all the messages on which m causally depends ($lastrcvd_j[k] \geq M[k,j]$ for all k), and there is no message destined for h_d on which m causally depends which is yet to be delivered to h_d ($\nexists \langle m', M', seqno'\rangle$ destined for h_d sent by S_k yet to be delivered such that $seqno' \leq M[k,j]$). If so, S_j transmits m to h_d. If m is not currently deliverable, it is kept in $rcvQ_j$ until it becomes deliverable. Like *YHH*, we do not update M_d immediately after delivering m to h_d, but we store m in $ackQ_d$. When h_d receives m, it sends back an acknowledge message, denoted by $ack(m)$, to S_j. On receiving $ack(m)$, S_j sets $M_d[i,j]$ to the maximum of its original value and $seqno$ (piggybacked on m). Then it sets each element in M_d to the maximum of its original value and the value of the corresponding element in M (also piggybacked on m). This prevents any outgoing message from h_d to become causally dependent on m that is sent before m is received by h_d. Figure 6 gives a more detailed description of the static module.

4.2 Handoff Module

In order to ensure causally ordered message delivery, some steps have to taken during handoff after an MH moves from one cell to another. We now describe the handoff module. Each MH h_l maintains a *mobility number*, mbl_l, which is initially set to 0. It is incremented every time a mobile host moves. Intuitively, mbl_l denotes the number of times h_l has changed cell. In addition, every MSS maintains an array of 2-tuples, denoted by *cell*, with an entry for each MH. The l^{th} entry of $cell_i$, $cell_i[l]$, is a 2-tuple $\langle mbl, mss\rangle$, where the value of $cell_i[l].mss$ represents S_i's knowledge of the location of h_l and the value of $cell_i[l].mbl$ indicates how "current" the knowledge is.

Consider a scenario when an MH h_l moves from the cell of S_i to the cell of S_j. After switching cell, h_l increments mbl_l and sends $register(mbl_l, S_i)$ message to S_j to inform S_j of its presence. Also, h_l retransmits the messages to S_j for which it did not receive the acknowledge message from its previous MSS S_i. On

S_i ::

var
$rcvQ$: queue of messages, initially ϕ;
$cell$: array $[1..n_h]$ of 2-tuples $\langle mbl, mss\rangle$, initially $[\langle 0, S_0^k\rangle]_{1\le k\le n_h}$;
$lastsent$, $lastrcvd$: array$[1..n_s]$ of integers, initially $\mathbf{0}$;
M : set of matrices $(n_s \times n_s)$, $(\{M_k \mid h_k \in \mathcal{H}_i\})$, each initially $\mathbf{0}$;
$ackQ$: set of FIFO queues of messages, $(\{ackQ_k \mid h_k \in \mathcal{H}_i\})$, each initially ϕ;
$sndQ$: set of FIFO queues of messages, $(\{sndQ_k \mid h_k \in \mathcal{H}_i\})$, each initially ϕ;
$canSend$: set of boolean variables, $(\{canSend_k \mid h_k \in \mathcal{H}_i\})$, each initially **true**;
$canDeliver$: set of boolean variables, $(\{canDeliver_k \mid h_k \in \mathcal{H}_i\})$, each initially **true**;

(A1) On receiving a data message m from h_s;
send an acknowledgement to h_s;
put m in $sndQ_s$;
call $process_sndQ(h_s)$;

(A2) On calling $process_sndQ(h_s)$;
if $(canSend_s)$ then
while $(sndQ \neq \phi)$ do
remove m from the head of $sndQ_s$;
let m be destined for h_d and S_j be $cell[d].mss$;
$lastsent[j]++$;
send $\langle m, M_s, lastsent[j]\rangle$ to S_j;
$M_s[i,j] := lastsent[j]$;
endwhile;
endif;

(A3) On receiving $\langle m, M, seqno\rangle$ from S_j;
$lastrcvd[j] := seqno$;
put $\langle m, M, seqno\rangle$ in $rcvQ$;
call $process_rcvQ()$;

(A4) On calling $process_rcvQ$;
repeat
forall $\langle m, M, seqno\rangle \in rcvQ$ do
let m be destined for h_d;
if ($canDeliver_d \wedge \langle \forall k :: lastrcvd[k] \ge M[k,i]\rangle \wedge$
$\langle \nexists\ \langle m', M', seqno'\rangle \in rcvQ :: (S_k$ sent m' for $h_d) \wedge (seqno' \le M[k,i])\rangle$) then
remove $\langle m, M, seqno\rangle$ from $rcvQ$;
call $deliver(\langle m, M, seqno\rangle)$;
endif;
endforall;
until $(rcvQ = \phi) \vee$ (no more messages can be delivered);

(A5) On calling $deliver(\langle m, M, seqno\rangle)$;
let m be destined for h_d;
put $\langle m, M, seqno\rangle$ in $ackQ_d$;
send m to h_d;

(A6) On receiving an acknowledgement from h_d;
remove $\langle m, M, seqno\rangle$ from the head of $ackQ_d$ and let S_j sent m;
$M_d[j,i] := \max\{M_d[j,i], seqno\}$;
$M_d := \max\{M_d, M\}$;

Figure 6: The static module for a mobile support station S_i

receiving this message h_l, S_j updates $cell_j[l]$ (its local knowledge about the location of h_l) and sends $handoff_begin(h_l, mbl_l)$ message to S_i. The MSS S_i, on receiving $handoff_begin(h_l, mbl_l)$ message, updates $cell_i[l]$ and sends $enable(h_l, M_l, ackQ_k)$ message to S_j. It then broadcasts $notify(h_l, mbl_l, S_j)$ message to all MSSs (except S_i and S_j), and waits for $last(h_l)$ message from all the MSSs to which it sent *notify*

S_i ::

var
$noOfLast$: set of integers, ($\{noOfLast_k \mid h_k \in \mathcal{H}_i\}$), each initially 0;
$handoffOver$: set of boolean variables, ($\{handoffOver_k \mid h_k \in \mathcal{H}_i\}$), each initially **true**;
$handoffQ$: set of priority queue of messages, ($\{handoffQ_k \mid h_k \in \mathcal{H}\}$), each initially ϕ;

(A7) On receiving $\langle register, mbl, S_j \rangle$ from h_l;
put $\langle register, mbl, S_j \rangle$ in $handoffQ_l$ using mbl as the key;
call $process_handoffQ(h_l)$;

(A8) On receiving $\langle handoff_begin, h_l, mbl \rangle$ from S_j;
put $\langle handoff_begin, mbl, S_j \rangle$ in $handoffQ_l$ using mbl as the key;
call $process_handoffQ(h_l)$;

(A9) On receiving $\langle notify, h_l, mbl, S_n \rangle$ from S_j;
if ($cell[l].mbl < mbl$) then $cell[l] := \langle mbl, S_n \rangle$;
send $\langle last, h_l \rangle$ to S_j;
call $process_handoffQ(h_l)$;

(A10) On receiving $\langle enable, h_l, M', ackQ', up_cell \rangle$;
forall $\langle h_k, mbl, S_n \rangle \in up_cell$ do
if ($cell[k].mbl < mbl$) then $cell[k] := \langle mbl, S_n \rangle$;
endforall;
$M_l := M'$;
while ($ackQ' \neq \phi$) do
remove $\langle m, M, seqno \rangle$ from the head of $ackQ'$ and let S_j sent m;
put $\langle m, M, seqno \rangle$ in $ackQ_l$;
send m to h_l;
$M_l[j, i] := \max\{M_l[j, i], seqno\}$;
$M_l := \max\{M_l, M\}$;
endwhile;
$canSend_l := \mathbf{true}$;
call $process_sndQ(h_l)$;

(A11) On receiving $\langle last, h_l \rangle$;
$noOfLast_l + +$;
if ($noOfLast_l = n_s - 2$) then
$canDeliver_l := \mathbf{false}$;
send $\langle handoff_over, h_l \rangle$ to $cell[l].mss$;
remove h_l from $\mathcal{H}_i$;
call $process_handoffQ(h_l)$;
endif;

(A12) On receiving $\langle handoff_over, h_l \rangle$;
$canDeliver_l := \mathbf{true}$;
$handoffOver_l := \mathbf{true}$;
$process_handoffQ(h_l)$;
$process_rcvQ()$;

Figure 7: The handoff module for a mobile support station S_i

message. Meanwhile, if any message received by S_i for h_l becomes deliverable, S_i marks it as "old" and forwards it to S_j.

On receiving $enable(h_l, M_l, ackQ_l)$ message from S_i, S_j first delivers all the messages in $ackQ_l$. It also updates M_l assuming all the messages in $ackQ_l$ have been received at h_l. Then S_j starts sending the application messages on behalf of h_l. S_j also delivers all the messages for h_l that are marked "old" in the order in which the messages arrived. However, messages destined for h_l that are not marked "old" are queued in $rcvQ_j$.

An MSS S_k, on receiving $notify(h_l, mbl_l, S_l)$ message, updates $cell_k[l]$ and then sends $last(h_l)$ message

S_i ::

(A13) On calling $process_handoffQ(h_l)$;
  let $\langle type, mbl, S_j \rangle$ be at the head of $handoffQ_l$;
  if $((type = register) \wedge (mbl = cell[l].mbl + 1) \wedge (h_l \notin \mathcal{H}_i))$ then
    remove the message from the head of $handoffQ_l$;
    add h_l to $\mathcal{H}_i$;
    $cell[l] := \langle mbl, S_i \rangle$;
    $canSend_l :=$ **false**;
    $canDeliver_l :=$ **false**;
    $handoffOver_l :=$ **false**;
    send $\langle handoff_begin, h_l, mbl \rangle$ to S_j;
  else if $((type = handoff_begin) \wedge (mbl = cell[l].mbl + 1) \wedge handoffOver_l)$ then
    remove the message from the head of $handoffQ_l$;
    $cell[l] := \langle mbl, S_j \rangle$;
    let up_cell be $\{\langle h_k, cell[k].mbl, cell[k].mss \rangle \mid h_k$ has changed cell since up_cell was last sent to $S_j\}$;
    send $\langle enable, h_l, M_l, ackQ_l, up_cell \rangle$ to S_j;
    broadcast $\langle notify, h_l, mbl, S_j \rangle$ to $\mathcal{S} \setminus \{S_i, S_j\}$;
  endif;

(A14) On receiving $\langle m, M, seqno, old, up_cell \rangle$;
  forall $\langle h_k, mbl, S_n \rangle \in up_cell$ do
    if $(cell[k].mbl < mbl)$ then $cell[k] := \langle mbl, S_n \rangle$;
  endforall;
  call $deliver(\langle m, M, seqno \rangle)$;

Figure 8: The handoff module for a mobile support station S_i (contd.)

to S_i. Observe that since the channels among all the MSSs are assumed to be FIFO, after S_i receives $last(h_l)$ message from S_k there are no messages in transition destined for h_l that are sent by S_k to S_i. On receiving $last(h_l)$ message from all the MSSs (to which *notify* message was sent), S_i sends $handoff_over(h_l)$ message to S_j. The handoff terminates at S_j after S_j receives $handoff_over(h_l)$ message. S_j can now start delivering messages to h_l. Meanwhile, if S_j receives $handoff_begin(h_l)$ message from some other MSS before the current handoff terminates, S_j responds to the message only after the handoff terminates.

Since we do not assume that the messages in the wired network are causally ordered, it is possible that a message m destined for h_l is sent to S_i (the old MSS of h_l), whereas its causally preceding message m', also destined for h_l, is sent to S_j (the new MSS of h_l). In order to prevent this, an MSS piggybacks additional information on all the message that contain application messages: messages destined for an MH (may or may not be marked as "old") and *enable* messages. On these messages, an MSS piggybacks its local knowledge of the location of all the mobile hosts that have changed their cells since it last communicated with the other MSS. On receiving this information, the other MSS updates its knowledge of the location of the MHs (its *cell*) based on their mobility number. In the worst case, this extra overhead could be as large as $O(n_h)$. In practice, we expect it to be much smaller. Let t_{snd} denote the mean inter-message generation time and t_{mov} be the mean inter-switch time for a MH. Then, the average extra overhead for uniform communication pattern (every MH has equal probability of sending a message to every other MH) is $\approx O(\frac{t_{snd}}{t_{mov}} n_s^2)$.

Our handoff module is more efficient than the handoff module in $AV2$ and $AV3$ since we do not require the messages exchanged among the MSSs to be causally ordered. Figure 7 and Figure 8 give a more detailed description of the handoff module. Figure 9 gives the modifications in the static module to incorporate mobile hosts.

Although we do not mention here but the mobility number has several usages. For instance, the messages exchanged between a MH and its MSS can also be tagged with the mobility number of the MH. It can

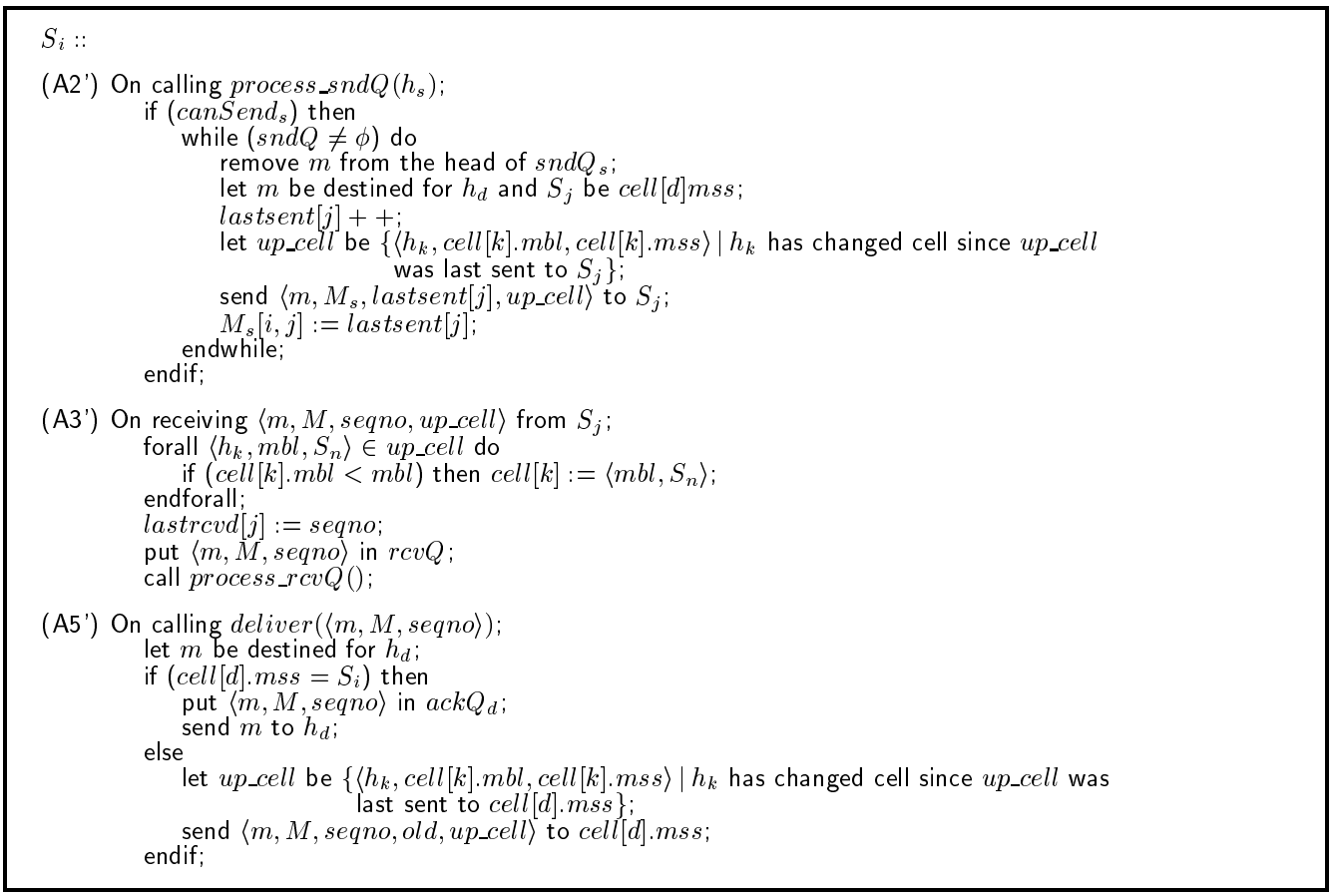


Figure 9: The modification in static module in presence of host movement in mobile support station S_i

then be used by an MSS to ignore messages received from an MH after it has sent *enable* message for that MH to the new MSS. It can also be used to correctly serialize the handoff procedures for an MH.

4.3 Proof of Correctness

We assume that a message sent to itself is immediately received by the sending process. Also, *enable*, *notify*, *last* and *handoff_over* messages are delivered as soon as they are received. The *register* and *handoff_begin* messages are delivered once the corresponding "if" condition is satisfied in (A13). Since the MSS does all the processing therefore for an application message m, $m.rcv = m.dlv$.

Note that since MHs are mobile and can change their cell, an application message m can be delivered to an MH by an MSS other than $\hat{m}.dst$ (either when m is received as on "old" message or m is in the acknowledgement queue of an *enable* message). In fact, m can be received multiple times by its destination MH. We consider m to be received (delivered) when the destination MH receives (delivers) it for the first time. Moreover, an application message can be sent multiple times (due to retransmission by the mobile host on failure to receive acknowledgement). We treat the retransmitted application message as a different application message and ignore the application message sent by an MH that is lost when the MH switched its cell. Here we assume that an MH can detect duplicate application messages and discard them. In our protocol, apart from the application message m, $\hat{m}$ also contains a matrix, denoted by $\hat{m}.M$, and a sequence number, denoted by $\hat{m}.seqno$. For convenience, $m.M = \hat{m}.M$ and $m.seqno = \hat{m}.seqno$. A

matrix M_i is less than or equal to M_j, denote by $M_i \preceq M_j$, iff $\langle \forall\, k, l :: M_i[k,l] \leq M_j[k,l] \rangle$.

Let e be an event on an MSS S_i. We use $mbl(e)$ and $lastrcvd(e)$ to denote the value of the vectors $cell[1:n_h].mbl$ and $lastrcvd$ respectively at S_i on occurrence of e. The k^{th} entry of the vector v is denoted by $v.k$. A vector v_i is less than or equal to a vector v_j, denoted by $v_i \preceq v_j$, iff $\langle \forall\, k :: v_i.k \leq v_j.k \rangle$. We use the same operator $\preceq$ to compare the vectors and the matrices. Intuitively, the value $cell[k].mss$ at S_i represents S_i's knowledge of the location of h_k and $cell[k].mbl$ indicates how "recent" the knowledge is. For a message m sent by an MSS, $mbl(m) = mbl(m.snd)$. For an application message m, $mbl(m) = mbl(\hat{m})$. Since for all k, $cell[k].mbl$ is monotonically non-decreasing for every MSS therefore $e \prec_s f \;\Rightarrow\; mbl(e) \preceq mbl(f)$.

The following lemmas and theorems prove the correctness of both the static and the handoff modules. The organization of the proof is as follows. We first prove Lemma 2-4 that are used in the proof of liveness and safety properties. Theorem 9 establishes the *liveness* property of the protocol, namely a message sent to a mobile host is eventually delivered. We prove the liveness property in two stages. We first prove that a message $\hat{m}$, carrying the application message m, is eventually delivered at its destination MSS (when m becomes deliverable at $\hat{m}.dst$ or $deliver(\hat{m})$ is called at $\hat{m}.dst$), $\hat{m}.dst$ (Lemma 8). Using it, we prove that the application message m is eventually delivered at its destination MH, $m.dst$. Theorem 13 establishes the *safety* property of the protocol, namely the modules implement causal ordering among mobile hosts.

In the lemmas that follow, let m_i and m_j be arbitrary application messages. Let $m_i.src = h_s$ and $m_i.dst = h_d$, and $m_j.src = h_{s'}$ and $m_j.dst = h_{d'}$. Let $\hat{m}_i.src = S_u$ and $\hat{m}_i.dst = S_v$, and $\hat{m}_i.src = S_{u'}$ and $\hat{m}_j.dst = S_{v'}$. For convenience, let $mbl(m_i).d = r$ and $mbl(m_j).d' = r'$. Note that $S_r^d = \hat{m}_i.dst = S_v$ and $S_{r'}^{d'} = \hat{m}_j.dst = S_{v'}$. Although, both S_v and S_r^d represent identical MSS $\hat{m}_i.dst$), we use S_r^d when we want to assert that S_v is the $r+1^{th}$ MSS of h_d and argue about the properties that hold during that time period. This usage is not limited to m_i.

4.3.1 Preliminary Lemmas

Let $\twoheadrightarrow_s$ denote the Lamport's happened before relation in the concrete view with respect to the messages on which *up_cell* is piggybacked. Observe that $\twoheadrightarrow_s \subseteq \rightarrow_s$. Let $enable(S_p^l)$ denote the enable message sent by S_p^l to S_{p+1}^l on processing *handoff_begin* message from S_{p+1}^l in the handoff module for h_l when h_l moves from the cell of S_p^l to the cell of S_{p+1}^l. Also, let $enable(S_p^l).M$ denote the matrix, M_l, piggybacked on the *enable* message. Since S_p^l does not process the *handoff_begin* message from S_{p+1}^l until it receives the *handoff_over* message from S_{p-1}^l and the protocol piggybacks *up_cell* on an *enable* message therefore,

(P 4.1) $$p < q \;\Rightarrow\; enable(S_p^l).snd \twoheadrightarrow_s enable(S_q^l).snd$$

Also, since M_l is monotonically non-decreasing, we have

(P 4.2) $$p \leq q \;\Rightarrow\; enable(S_p^l).M \preceq enable(S_q^l).M$$

Lemma 2 $m_i.snd \rightarrow_h m_j.snd \;\Rightarrow\; m_i.seqno \leq m_j.M[u,v]$

Proof: The proof is by induction on the number of messages, n, involved in the smallest causal chain (with respect to $\rightarrow_h$) from $m_i.snd$ to $m_j.snd$. Let $\hat{m}_i.src = S_a^s$ and $\hat{m}_j.src = S_{a'}^{s'}$, where $a = mbl(m_i).s$ and $a' = mbl(m_j).s'$. Note that $S_u = S_a^s$ and $S_{u'} = S_{a'}^{s'}$.

Base Case ($n = 0$): In this case, $m_i.snd \prec_h m_j.snd$. Observe that $h_s = h_{s'}$ and $a' \geq a$. There are two cases to consider depending on whether h_s switched its cell after sending m_i.

Case 1 $[a' = a]$: It can be verified from the protocol that as soon as $\hat{m}_i$ is sent, M_s is updated (A2'). Using monotonicity of M_s, we have $m_i.seqno \leq m_j.M[u,v]$.

Case 2 $[a' > a]$: Since S_a^s does not send forward any message on behalf of h_s to any MSS after sending the *enable* message, therefore $\hat{m}_i.snd \prec_s enable(S_a^s).snd$. Also, $S_{a'}^s$ does not forward any message on behalf of h_s until it receives $enable(S_{a'-1}^s)$, therefore $enable(S_{a'-1}^s).dlv \prec_s \hat{m}_j.snd$. Using P 4.2, monotonicity of M_s and $a' - 1 \geq a$, we get,

$$m_i.seqno \leq enable(S_a^s).M[u,v] \leq enable(S_{a'-1}^s).M[u,v] \leq m_j.M[u,v]$$

Thus, in any case, $m_i.seqno \leq m_j.M[u,v]$.

Induction Step $(n \geq 1)$: Let m_k denote the last message in the smallest causal chain (with respect to $\rightarrow_s$) from $m_i.snd$ to $m_j.snd$. Then, by induction $m_i.seqno \leq m_k.M[u,v]$. Let $S_b^{s'}$ be the MSS which first delivered m_k to $h_{s'}$. Observe that $a' \geq b$. Let $ack(m_k)$ denote the acknowledgement message sent by $h_{s'}$ on receiving m_k. There are two cases to consider depending on whether $h_{s'}$ switched its cell after receiving (or delivering) m_k.

Case 1 $[a' = b]$: Since $m_k.dlv \prec_h m_i.snd$, therefore $ack(m_k)$ is received at $S_{a'}^{s'}$ before m_i. Moreover, when $ack(m_k)$ is received, m_k is at the head of $ackQ_{s'}$ (channel between an MH and its MSS is reliable and FIFO). On receiving $ack(m_k)$, $S_{a'}^{s'}$ updates $M_{s'}$ to reflect the "delivery" of m_k at $h_{s'}$ which involves taking component-wise maximum of $m_k.M$ and $M_{s'}$. Using monotonicity of $M_{s'}$, we have $m_k.M[u,v] \leq m_j.M[u,v]$.

Case 2 $[a' > b]$: Due to the movement of $h_{s'}$, it is possible that although $h_{s'}$ received m_k and sent $ack(m_k)$ to $S_b^{s'}$, $S_b^{s'}$ did not receive $ack(m_k)$ before it sends the *enable* message. Therefore, on receiving the *enable* message from $S_b^{s'}$, $S_{b+1}^{s'}$ updates $M_{s'}$ assuming that all the messages in $ackQ_{s'}$ have been received at $h_{s'}$ before proceeding further. Using P 4.2 and monotonicity of $M_{s'}$, we have $m_k.M[u,v] \leq m_j.M[u,v]$.

Thus, in any case, $m_i.seqno \leq m_j.M[u,v]$. Therefore by induction the lemma holds. ■

Lemma 3 $m_i.seqno \leq m_j.M[u,v] \;\Rightarrow\; \hat{m}_i.snd \twoheadrightarrow_s \hat{m}_j.snd$

Proof: Assume $m_i.seqno \leq m_j.M[u,v]$. The proof is by construction. We first prove the following property satisfied by m_i and m_j,

$$\begin{aligned} m_i.seqno \leq m_j.M[u,v] \;\Rightarrow\; & (\hat{m}_i.snd \twoheadrightarrow_s \hat{m}_j.snd) \vee \\ & \langle \exists\, m_k :: (m_i.seqno \leq m_k.M[u,v]) \wedge (\hat{m}_k.snd \twoheadrightarrow_s \hat{m}_j.snd) \rangle \end{aligned}$$

Let $\hat{m}_j.src = S_{a'}^{s'}$ where $a' = mbl(m_j).s'$. Observe that $m_i.seqno \geq 1$ and $M_{s'}$ is initially $\mathbf{0}$. Since $M_{s'}$ is monotonically non-decreasing, therefore there exists an MSS where $M_{s'}$ was updated which made the inequality true. Let $S_b^{s'}$ be the first such MSS in the sequence $\{S_l^{s'}\}$, and e_k be the earliest event on it such that the inequality holds just after e_k. Note that $a' \geq b$ and $M_{s'}$ is updated only either due to a message sent by $h_{s'}$ or due to a message destined for $h_{s'}$. Let m_k denote the message involved in e_k. Observe that $m_k \neq m_j$. In the former case (the inequality became true due to a message sent by $h_{s'}$), $\hat{m}_k.src = S_u$ and $\hat{m}_k.dst = S_v$. Since *lastsent* on $S_b^{s'}$ is monotonically non-decreasing and $m_i.seqno \leq m_k.seqno$, therefore either $m_i = m_k$ or $\hat{m}_i.snd \prec_s \hat{m}_k.snd$. Moreover, if $a' = b$ then $\hat{m}_k.snd \prec_s \hat{m}_j.snd$, otherwise $\hat{m}_k.snd \prec_s enable(S_b^{s'}).snd$ and $enable(S_{a'-1}^{s'}).dlv \prec_s \hat{m}_j.snd$. Using P 4.1, we have $\hat{m}_i.snd \twoheadrightarrow_s \hat{m}_j.snd$.

In the latter case, as before, if $a' = b$ then $e_k \prec_s \hat{m}_j.src$, otherwise $e_k \rightarrow\!\!\!\!\!\cdot_s \hat{m}_j.snd$ ($M_{s'}$ is not updated at $S_b^{s'}$ after it sends the *enable* message). Moreover, it can be verified from the protocol that $\hat{m}_k$ was in $ackQ_{s'}$ when $M_{s'}$ was updated. Let $\hat{m}_k.dst = S_c^{s'}$ where $c = mbl(m_k).s'$. Observe that $\hat{m}_k$ first enters $ackQ_{s'}$ either at $S_c^{s'}$ on occurrence of $\hat{m}_k.dlv$ or at $S_{c+1}^{s'}$ on being received as an "old" message. After that, it gets transferred to the next MSS piggybacked on the *enable* message. Since the messages containing application message, the messages tagged as "old" and the *enable* messages are piggybacked with *up_cell*, therefore $\hat{m}_k.snd \rightarrow\!\!\!\!\!\cdot_s e_k$. Thus, $\hat{m}_k.snd \rightarrow\!\!\!\!\!\cdot_s \hat{m}_j.snd$. There are again two cases to consider. The inequality became true either due to *seqno* of m_k or as a result of taking component-wise maximum of $m_k.M$ and $M_{s'}$. In the first case, $\hat{m}_k.src = S_u$, $\hat{m}_k.dst = S_v$ and $m_i.seqno \leq m_k.seqno$. Therefore, either $m_i = m_k$ or $\hat{m}_i.snd \prec_s \hat{m}_k.snd$. Combining with earlier result, we get $\hat{m}_i.snd \rightarrow\!\!\!\!\!\cdot_s \hat{m}_j.snd$. Finally, in the second case, $m_i.seqno \leq m_k.M[u,v]$.

Thus, $(\hat{m}_i.snd \rightarrow\!\!\!\!\!\cdot_s \hat{m}_j.snd) \vee \langle \exists\, m_k :: (m_i.seqno \leq m_k.M[u,v]) \wedge (\hat{m}_k.snd \rightarrow\!\!\!\!\!\cdot_s \hat{m}_j.snd) \rangle$ holds. We can apply the same property to m_i and m_k since $m_i.seqno \leq m_k.M[u,v]$. We claim that at most n_h applications of the property establishes $\hat{m}_i.snd \rightarrow\!\!\!\!\!\cdot_s \hat{m}_j.snd$. The proof is by contradiction. Assume the contrary. Then, there is a chain of messages, $m_{k_1}, m_{k_2}, \ldots m_{k_l}, m_j$ such that $\hat{m}_{k_1}.snd \rightarrow\!\!\!\!\!\cdot_s \hat{m}_j.snd$ ($\rightarrow\!\!\!\!\!\cdot_s$ is transitive) and $l > n_h$. Using the pigeon-hole principle, we can infer that at least two messages in the chain are sent by the same MH. Let the messages be m_{k_p} and m_{k_q}. Also, let e_{k_p} and e_{k_q} be the events used in the proof of the property. Since both events involve update of the MH matrix, therefore either $e_{k_p} \rightarrow_s e_{k_q}$ or $e_{k_q} \rightarrow_s e_{k_p}$ holds which contradicts the choice of e_{k_p} or e_{k_q}. Thus, the lemma holds. ■

Lemma 4 $\hat{m}_i.snd \rightarrow\!\!\!\!\!\cdot_s \hat{m}_j.snd \Rightarrow mbl(m_i) \preceq mbl(m_j)$

Proof: The lemma can be proved by induction on the number of messages, n, involved in smallest causal chain (with respect to $\rightarrow\!\!\!\!\!\cdot_s$) from $\hat{m}_i.snd$ to $\hat{m}_j.snd$. The proof is straightforward and is left to the reader. ■

4.3.2 Liveness Property

Lemma 5 *Every handoff procedure for a mobile host terminates.*

Proof: Let $handoff(l,p)$ denote the handoff procedure between S_p^l and S_{p+1}^l for h_l, when h_l moves from the cell of S_p^l to the cell of S_{p+1}^l. The lemma can be proved easily by induction on p, $0 \leq p < n(h_l)$. ■

Let $handoff_over(S_p^l)$ denote the *handoff_over* message sent by S_p^l to S_{p+1}^l in the handoff module for h_l, when h_l moves from the cell of S_p^l to the cell of S_{p+1}^l. Since S_p^l, does not process the *handoff_begin* message from S_{p+1}^l until it receives the *handoff_over* message from S_{p-1}^l, therefore we have,

(P 4.3) $$p < q \Rightarrow handoff_over(S_p^l).snd \rightarrow_s handoff_over(S_q^l).snd$$

Let $\hat{m}_i.ercvd$ denote the earliest event on S_v such that $\hat{m}_i.M[1:n_s,v] \preceq lastrcvd(\hat{m}_i.ercvd)$. Observe that $\langle \forall\, e : \hat{m}_i.ercvd \prec_s e : \hat{m}_i.M[1:n_s,v] \preceq lastrcvd(e) \rangle$ is true. Intuitively, $\hat{m}_i.ercvd$ represents the earliest event on S_v when all the messages sent to S_v on which m_i causally depends (potentially) have been received at S_v.

Lemma 6 *$\hat{m}_i.ercvd$ occurs eventually. Moreover, if S_r^d is not the final mobile support station for h_d, i.e. $r < n(h_d)$, then $\hat{m}_i.ercvd \prec_s handoff_over(S_r^d).snd$.*

Proof: Consider a message $\hat{m}_k$ destined for S_v such that $\hat{m}_k.seqno \leq \hat{m}_i.M[w,v]$, where $S_w = \hat{m}_k.src$. We claim that $\hat{m}_k$ is eventually received, i.e. $\hat{m}_k.rcv$ occurs eventually. Furthermore, if $r < n(h_d)$ then $\hat{m}_k.rcv \prec_s handoff_over(S_r^d).snd$. Assume S_r^d is the final MSS for h_d. Since the channels among MSSs are reliable, therefore $\hat{m}_k$ is received eventually. Otherwise, assume $r < n(h_d)$. We have three cases to consider depending on the source MSS of $\hat{m}_k$. Let S_n denote the MSS to whose cell h_d moves after leaving the cell of S_r^d. Let $handoff_begin(S_{r+1}^d)$ denote the *handoff_begin* message sent by S_{r+1}^d to S_r^d in the handoff procedure when h_d switches cell. Let $notify(S_r^d)$ represent the *notify* message broadcast by S_r^d to the MSSs in the handoff procedure and let $last(S_w, S_r^d)$ denote the corresponding *last* message sent by S_w to S_r^d.

Case 1 $[S_w = S_v]$: In this case, $\hat{m}_k.snd \prec_s handoff_begin(S_{r+1}^d).dlv$. Assume the contrary. After processing the *handoff_begin* message, the value of $cell[d].mlb$ at S_v becomes $r+1$. Thus, $mbl(\hat{m}_k).d > r$. Using Lemma 3 and Lemma 4, we can infer that $r < mbl(\hat{m}_i).d$, a contradiction. Since the messages sent to itself are received immediately and $handoff_begin(S_{r+1}^d).dlv \prec_s handoff_over(S_r^d).snd$, therefore $\hat{m}_k.rcv \prec_s handoff_over(S_r^d).snd$.

Case 2 $[S_w = S_n]$: In this case, $\hat{m}_k.snd \prec_s handoff_begin(S_{r+1}^d).snd$. The proof is identical to the proof in Case 1. Since the channels are reliable and FIFO, therefore $\hat{m}_k.rcv \prec_s handoff_begin(S_{r+1}^d).rcv$. Also, $handoff_begin(S_{r+1}^d).rcv \prec_s handoff_over(S_r^d).snd$. Thus, $\hat{m}_k.rcv \prec_s handoff_over(S_r^d).snd$.

Case 3 $[S_w \in \mathcal{S} \setminus \{S_v, S_n\}]$: Finally, in this case, $\hat{m}_k.snd \prec_s notify(S_r^d).dlv$. Since the channels are reliable and FIFO, and $notify(S_r^d).dlv \prec_s last(S_w, S_r^d).snd$, therefore $\hat{m}_k.rcv \prec_s last(S_w, S_r^d).rcv$. Also, $last(S_w, S_r^d).rcv \prec_s handoff_over(S_r^d).snd$. Thus, $\hat{m}_k.rcv \prec_s handoff_over(S_r^d).snd$.

In any case, $\hat{m}_k.rcv \prec_s handoff_over(S_r^d).snd$. Thus, for all $\hat{m}_k$ destined for S_v such that $\hat{m}_k.seqno \leq \hat{m}_i.M[w,v]$, where $\hat{m}_k.src = S_w$, we have $\hat{m}_k.rcv \prec_s handoff_over(S_r^d).snd$. Since as soon as a message is received *lastrcvd* is updated, therefore $\hat{m}_i.ercvd \prec_s handoff_over(S_r^d).snd$. Therefore the lemma holds. ∎

Lemma 7 *$\hat{m}_i.rcv$ occurs eventually. Moreover, if S_r^d is not the final mobile support station for h_d, i.e. $r < n(h_d)$, then $\hat{m}_i.rcv \prec_s handoff_over(S_r^d).snd$.*

Proof: The lemma can be proved by doing a case analysis identical to the one in Lemma 6. The proof is left to the reader. ∎

Lemma 8 *$\hat{m}_i$ is eventually delivered (at its destination mobile support station S_v). Moreover, if S_r^d is not the final mobile support station for h_d, i.e. $r < n(h_d)$, then $\hat{m}_i.dlv \prec_s handoff_over(S_r^d).snd$.*

Proof: Let $\mathcal{M}_A$ denote the set of messages which contain application messages (not tagged as "old") sent by a mobile support station to another mobile support station to be delivered to the destination mobile hosts. Let $\mathcal{M}_C$ be the set of messages on which *up_cell* is piggybacked. We first define a binary relation, $\sqsubset$, on $\mathcal{M}_A$ as follows,

$$\hat{m}_i \sqsubset \hat{m}_j \stackrel{def}{=} (h_d = h_{d'}) \wedge (S_v = S_{v'}) \wedge (\hat{m}_i.seqno \leq \hat{m}_j.M[u,v])$$

Observe that $\mathcal{M}_A \subseteq \mathcal{M}_C$ and $\sqsubset \subseteq \twoheadrightarrow_s$ (Lemma 3). Also, $(\mathcal{M}_C, \twoheadrightarrow_s)$ is a well-founded set. Thus, we can infer that $(\mathcal{M}_A, \sqsubset)$ is also a well-founded set. Let $\mathcal{P}.\hat{m}_k$ be "the lemma holds for $\hat{m}_k$". Assume $\langle \forall\, \hat{m}_k : \hat{m}_k \sqsubset \hat{m}_i : \mathcal{P}.\hat{m}_k \rangle$. There are two cases to consider: $r = n(h_d)$ or $r < n(h_d)$.

Case 1 [$r < n(h_d)$]: Using Lemma 7, we have $\hat{m}_i.rcv \prec_s handoff_over(S_r^d).snd$. If S_r^d is the initial MSS of h_d, i.e. ($r = 0$), then $canDeliver_d$ is true initially. Otherwise, using Lemma 5 we can infer that $handoff_over(S_{r-1}^d).dlv$ eventually occurs at S_r^d after which $canDeliver_d$ is set to true. Moreover, $canDeliver_d$ remains true until S_r^d sends the *handoff_over* message to S_{r+1}^d. Let *canDeliver* be the earliest event on S_r^d after which $canDeliver_d$ is true. Then $canDeliver \prec_s handoff_over(S_r^d).snd$. From Lemma 6, we can conclude that $\hat{m}_i.ercvd \prec_s handoff_over(S_r^d).snd$. Consider $\hat{m}_k$ such that $\hat{m}_k \sqsubset \hat{m}_i$. Using definition of $\sqsubset$, Lemma 3 and Lemma 4, we have $m_k.dst = h_d$ and $mbl(\hat{m}_k).d \leq mbl(\hat{m}_i).d = r$. Therefore, using induction hypothesis and P 4.3, we get $\hat{m}_k.dlv \prec_s handoff_over(S_r^d).snd$. Observe that after all messages $\hat{m}_k$ such that $\hat{m}_k \sqsubset \hat{m}_i$ have been delivered, then the last expression in the conjunct of the "if" condition in (A4) is never falsified. Let e be the latest of all the events in $\{\hat{m}_i.rcv, canDeliver, \hat{m}_i.ercvd\} \cup \{\hat{m}_k.dlv \mid \hat{m}_k \sqsubset \hat{m}_i\}$. Then $e \prec_s handoff_over(S_r^d).snd$. After e, the "if" condition in (A4) evaluates to true for $\hat{m}_i$, and $deliver(\hat{m}_i)$ is called. Thus, $\hat{m}_i.dlv \prec_s handoff_over(S_r^d).snd$. Therefore $\mathcal{P}.\hat{m}_i$ holds.

Case 2 [$r = n(h_d)$]: In this case, we have to prove that $\hat{m}_i.dlv$ eventually occurs. The proof is quite similar to but simpler than the proof for Case 1.

Hence by strong induction, the lemma holds. ■

Theorem 9 *(**liveness**) m_i is eventually delivered (at its destination mobile host, h_d).*

Proof: We first show that $\hat{m}_i$ eventually enters $ackQ_d$. If S_r^d is the final mobile support station for h_d or $\hat{m}_i.dlv \prec_s handoff_begin(S_r^d).dlv$ then $\hat{m}_i$ enters $acks_d$ as soon as $\hat{m}_i.dlv$ occurs. Otherwise, on occurrence of $\hat{m}_i.dlv$, $\hat{m}_i$ is sent to S_{r+1}^d where it is inserted into $ackQ_d$ on being received. Let S_t^d, $r \leq t \leq n(h_d)$ be the MSS such that h_d stays for sufficiently long time in the cell of S_t^d after $\hat{m}_i$ enters $ackQ_d$. Let $\mathcal{M}_{ack}$ be the set of messages that entered $ackQ_d$ at S_t^d (including messages that were already in $ackQ_d$ when $ackQ_d$ was transferred to S_t^d) before $\hat{m}_i$. Note that the messages are sent to h_d in the order in which they enter $ackQ_d$ ((A5') and (A10)). Moreover, after receiving $|\mathcal{M}_{ack}|$ acknowledgement messages from h_d, $\hat{m}_i$ will be at the front in $ackQ_d$. Since the channel between an MH and its MSS is reliable and FIFO, therefore S_t^d receives $|\mathcal{M}_{ack}|^{th}$ acknowledgement message from h_d if h_d does not switch cell for a sufficiently long time. Thus, m_i is delivered at h_d. ■

4.3.3 Safety Property

Lemma 10 *If $\hat{m}_i$ enters $ackQ_d$ before $\hat{m}_j$ then $m_i.dlv \prec_h m_j.dlv$.*

Proof: Note that $h_d = h_{d'}$. Let S_t^d and $S_{t'}^d$ denote the MSSs that delivered m_i and m_j respectively to h_d for the first time (t and t' exist due to Theorem 9). If $t < t'$ then it can be easily proved that $m_i.dlv \prec_h m_j.dlv$. Therefore, assume $t \geq t'$. Observe that in the protocol as soon as a message is inserted in $ackQ_d$ at $S_{t'}^d$, it is also dispatched to h_d ((A5') and (A10)). Thus, at $S_{t'}^d$, m_i is sent to h_d before m_j. Since the channel between an MH and its MSS is FIFO, therefore h_d receives m_i before m_j. Hence $m_i.dlv \prec_h m_j.dlv$. ■

Lemma 11 $mbl(m_i).d < mbl(m_j).d \;\Rightarrow\; \neg(m_j.dlv \prec_h m_i.dlv)$

Proof: If $h_d \neq h_{d'}$ then the consequent (and hence the lemma) is trivially true. Therefore assume $h_d = h_{d'}$. We first prove that $\hat{m}_i$ enters $ackQ_d$ before $\hat{m}_j$. If $\hat{m}_i.dlv \prec_s handoff_begin(S^d_{r+1}).dlv$ then $\hat{m}_i$ enters $ackQ_d$ at S^d_r. Otherwise, on occurrence of $\hat{m}_i.dlv$, $\hat{m}_i$ is sent to S^d_{r+1} where it is inserted into $ackQ_d$ as soon as it is received. Using Lemma 8 and the fact that the channels among MSSs are FIFO, we can infer that $\hat{m}_i$ is received at S^d_{r+1} before $handoff_over(S^d_r)$. Also, from the protocol we know that $\hat{m}_j$ cannot enter $ackQ_d$ before $handoff_over(S^d_{r'-1})$ is received. Thus, using P 4.3, we can conclude that in any case $\hat{m}_i$ enters $ackQ_d$ before $\hat{m}_j$. Finally, using Lemma 10, we have $m_i.dlv \prec_h m_j.dlv$. ■

Lemma 12 $(mbl(m_i).d = mbl(m_j).d) \wedge (m_i.snd \rightarrow_h m_j.snd) \;\Rightarrow\; \neg(m_j.dlv \prec_h m_i.dlv)$

Proof: If $h_d \neq h_{d'}$ then the consequent (and hence the lemma) is trivially true. Therefore assume $h_d = h_{d'}$. We first prove that $\hat{m}_i.dlv \prec_s \hat{m}_j.dlv$. Note that $S_v = S_{v'}$. From Lemma 2, we can conclude that $\hat{m}_i.seqno \leq \hat{m}_j.M[u,v]$. Observe that $\hat{m}_j.ercvd$ cannot occur before $\hat{m}_i.rcv$. Moreover, after $\hat{m}_j.ercvd$ occurs, $\hat{m}_j$ cannot be delivered until $\hat{m}_i$ is delivered. Thus, $\hat{m}_i.dlv \prec_s \hat{m}_j.dlv$. If S^d_r is the final MSS for h_d then as soon as $\hat{m}.dlv$ occurs it is inserted into $ackQ_d$. Therefore, $\hat{m}_i$ is inserted into $ackQ_d$ before $\hat{m}_j$. Otherwise, there are three cases to consider:

Case 1 $[\hat{m}_i.dlv \prec_s \hat{m}_j.dlv \prec_s handoff_begin(S^d_{r+1}).dlv]$: On occurrence of $\hat{m}_i.dlv$ ($\hat{m}_j.dlv$), $\hat{m}_i$ ($\hat{m}_j$) is inserted into $ackQ_d$. Hence $\hat{m}_i$ enters $ackQ_d$ before $\hat{m}_j$.

Case 2 $[\hat{m}_i.dlv \prec_s handoff_begin(S^d_{r+1}).dlv \prec_s \hat{m}_j.dlv]$: On occurrence of $\hat{m}_i.dlv$, $\hat{m}_i$ is inserted into $ackQ_d$. On processing $handoff_begin(S^d_{r+1})$, $ackQ_d$ is piggybacked on the $enable(S^d_r)$ message and sent to S^d_{r+1}. Then, when $\hat{m}_j.dlv$ occurs, $\hat{m}_j$ is sent to S^d_{r+1} where it enters $ackQ_d$. Since the channels among MSSs are reliable and FIFO, therefore $\hat{m}_i$ enters $ackQ_d$ before $\hat{m}_j$.

Case 3 $[handoff_begin(S^d_{r+1}).dlv \prec_s \hat{m}_i.dlv \prec_s \hat{m}_j.dlv]$: On occurrence of $\hat{m}_i.dlv$ ($\hat{m}_j.dlv$), $\hat{m}_i$ ($\hat{m}_j$ is sent to S^d_{r+1} tagged as on "old" message. On receiving $\hat{m}_i$ ($\hat{m}_j$), S^d_{r+1} inserts $\hat{m}_i$ ($\hat{m}_j$) into $ackQ_d$. Since the channels among MSSs are reliable and FIFO, therefore $\hat{m}_i$ enters $ackQ_d$ before $\hat{m}_j$.

In any case, $\hat{m}_i$ enters $ackQ_d$ before $\hat{m}_j$. Finally, using Lemma 10, we have $m_i.dlv \prec_h m_j.dlv$. ■

Theorem 13 *The protocol implements causal ordering among mobile hosts. In other words,*

$$m_i.snd \rightarrow_h m_j.snd \;\Rightarrow\; \neg(m_j.dlv \prec_h m_i.dlv)$$

Proof: The proof is a straightforward manipulation of the lemmas.

$$\begin{aligned}
& m_i.snd \rightarrow_h m_j.snd \\
\Rightarrow\; & m_i.seqno \leq m_j.M[u,v] && \text{; Lemma 2} \\
\Rightarrow\; & \hat{m}_i.snd \twoheadrightarrow_s \hat{m}_j.snd && \text{; Lemma 3} \\
\Rightarrow\; & mbl(m_i) \preceq mbl(m_j) && \text{; Lemma 4} \\
\Rightarrow\; & mbl(m_i).d \leq mbl(m_j).d && \text{; definition of } \preceq\text{, instantiation} \\
\equiv\; & (mbl(m_i).d < mbl(m_j).d) \vee (mbl(m_i).d = mbl(m_j).d) && \text{; definition of } \leq
\end{aligned}$$

$\Rightarrow$ $(mbl(m_i).d < mbl(m_j).d) \vee ((mbl(m_i).d = mbl(m_j).d) \wedge (m_i.snd \rightarrow_h m_j.snd))$

; use antecedent

$\Rightarrow$ $\neg(m_j.dlv \prec_h m_i.dlv) \vee \neg(m_j.dlv \prec_h m_i.dlv)$; Lemma 11, Lemma 12

$\Rightarrow$ $\neg(m_j.dlv \prec_h m_i.dlv)$; idempotence of $\vee$

Thus, the theorem holds. ■

4.4 Characterization of Static Module

In this section we state and prove the predicate that characterizes our static module. The static module in Section 4.1 implements,

$$(\mathbf{CO''}) \quad \langle \exists\, m_k : \hat{m}_i.dst = \hat{m}_k.dst : (\hat{m}_i.snd \preccurlyeq_s \hat{m}_k.snd) \wedge (m_k.snd \rightarrow_h m_j.snd) \rangle \Rightarrow \neg(m_j.dlv \prec_h m_i.dlv) \wedge \neg(\hat{m}_j.dlv \prec_s \hat{m}_i.rcv),$$

where $e \preccurlyeq_s f$ iff $(e = f) \vee (e \prec_s f)$, under the assumption that the channels among MSSs are FIFO. Moreover, if the channels among MSSs are not FIFO then it implements,

$$CO'' \wedge (\hat{m}_i.snd \prec_s \hat{m}_j.snd \Rightarrow \neg(\hat{m}_j.dlv \prec_s \hat{m}_i.rcv))$$

For convenience, let $FO'' \stackrel{def}{=} \hat{m}_i.snd \prec_s \hat{m}_j.snd \Rightarrow \neg(\hat{m}_j.dlv \prec_s \hat{m}_i.rcv)$. For the following proofs, we define $m_i.P$ for an application message m_i as follows,

$$m_i.P[u,v] = \max\{\{m_k \mid (\hat{m}_k.src = S_u) \wedge (\hat{m}_k.dst = S_v) \wedge (m_k.snd \rightarrow_h m_i.snd)\}\},$$

where $\max\{S\}$ returns the message with the largest *seqno* in the set S. Also, $\max\{\phi\} = \perp$, where $\perp.seqno = 0$ and $\perp \rightarrow_h m_i$.

Lemma 14 *For an application message m_i, $m_i.M[u,v] = m_i.P[u,v].seqno$ for all u and v.*

Proof: Using Lemma 2 and definition of $m_i.P[u,v]$, we can infer that $m_i.P[u,v].seqno \leq m_i.M[u,v]$ (the inequality trivially holds if $m_i.P[u,v] = \perp$). Assume $m_i.M[u,v] > m_i.P[u,v].seqno$. We will derive a contradiction. Let $m_i.M[u,v] = n, n > 0$. We first prove the following property for the application message m_i,

$$m_i.M[u,v] = n \Rightarrow \langle \exists m_k :: (((\hat{m}_k.src = S_u) \wedge (\hat{m}_k.dst = S_v) \wedge (m_k.seqno = n)) \vee (m_k.M[u,v] = n)) \wedge (m_k.snd \rightarrow_h m_i.snd) \rangle$$

Let $m_i.src = h_s$. Observe that $n > 0$ and M_s is initially $\mathbf{0}$. Since M_s is monotonically non-decreasing, therefore there exists an event on $\hat{m}_i.src$ when M_s was updated which made the equality, $M_s[u,v] = n$, true. Let e_k be the earliest event on it such that the equality holds just after e_k. Note that M_s is updated only either due to a message sent by h_s or due to a message received by h_s. Let m_k denote the application message involved in e_k. Observe that $e_k \prec_s \hat{m}_i.snd$. In the former case (the inequality became true due to a message sent by h_s), $\hat{m}_k.src = S_u$ and $\hat{m}_k.dst = S_v$. Moreover, $m_k.seqno = n$ and $m_k.snd \prec_h m_i.snd$. In the latter case, there are again two cases to consider. The equality became true either due to *seqno* of m_k or as a result of taking component-wise maximum of $m_k.M$ and M_s. In the

first case, $\hat{m}_k.src = S_u$, $\hat{m}_k.dst = S_v$ and $m_k.seqno = n$. In the second case $m_k.M[u,v] = n$. Moreover, in both cases, $m_k.snd \rightarrow_h m_i.snd$.

Thus, the property holds. If the second term of the "$\vee$" expression holds for m_k then we can apply the same argument since in that case $m_k.M[u,v] = n, n > 0$. We claim that at most n_h applications of the property establishes $\langle \exists m_k :: (\hat{m}_k.src = S_u) \wedge (\hat{m}_k.dst = S_v) \wedge (m_k.seqno = n) \wedge (m_k.snd \rightarrow_h m_i.snd) \rangle$. The proof is by contradiction. Assume the contrary. Then, there is a chain of messages, $m_{k_1}, m_{k_2}, \ldots m_{k_l}, m_i$ such that $m_{k_1}.snd \rightarrow_h m_i.snd$ ($\rightarrow_h$ is transitive) and $l > n_h$. Using the pigeon-hole principle, we can infer that at least two messages in the chain are sent by the same mobile host. Let the messages be m_{k_p} and m_{k_q}. Also, let e_{k_p} and e_{k_q} be the events used in the proof of the property. Then $e_{k_p} \rightarrow_s e_{k_q}$ or $e_{k_q} \rightarrow_s e_{k_p}$ holds which contradicts the choice of e_{k_p} or e_{k_q}. Thus, there exists an application message m_k such that $\hat{m}_k.src = S_u$, $\hat{m}_k.dst = S_v$, $m_k.seqno = n$ and $m_k.snd \rightarrow_h m_i.snd$. Also, $m_i.M[u,v] = n = m_k.seqno > m_i.P[u,v].seqno$ which contradicts the definition of $m_i.P[u,v]$. Hence $m_i.M[u,v] = m_i.P[u,v].seqno$ and the lemma holds. ■

Lemma 15 *For any two application messages m_i and m_j such that $\hat{m}_i.src = S_u$ and $\hat{m}_i.dst = S_v$, the static module satisfies,*

$$\langle \exists\, m_k : \hat{m}_i.dst = \hat{m}_k.dst : (\hat{m}_i.snd \preccurlyeq_s \hat{m}_k.snd) \wedge (m_k.snd \rightarrow_h m_j.snd) \rangle \;\equiv\; m_i.seqno \leq m_j.M[u,v]$$

Proof:
($\Rightarrow$)

(A.1) $m_i.snd \rightarrow_h m_j.snd \;\Rightarrow\; m_i.seqno \leq m_j.M[u,v]$
We prove (A.1) by induction on the number of messages,n, in the causal chain (with respect to $\rightarrow_h$) from $m_i.snd$ to $m_j.snd$.

Base Case ($n = 0$): In this case, $m_i.snd \prec_h m_j.snd$. On sending $\hat{m}_i$, S_u sets the $(u,v)^{th}$ entry of the host matrix to $m_i.seqno$. Since the wireless channels are FIFO and the host matrix is monotonically non-decreasing, therefore $m_i.seqno \leq m_j.M[u,v]$.

Induction Step ($n > 0$): Let $m_j.src = h_{s'}$. Let m_l be the last message in the causal chain. Using induction, we get $m_i.seqno \leq m_l.M[u,v]$. Observe that m_l is delivered to $h_{s'}$ before $m_j.snd$ occurs (to create the causal dependency). Since wireless channels are FIFO and reliable therefore acknowledge message for m_l, $ack(m_l)$, is received before m_j. On receiving $ack(m_l)$, $\hat{m}_j.src$ sets $M_{s'}$ to component-wise maximum of $m_l.M$ and $M_{s'}$. Hence, we have $m_l.M[u,v] \leq m_j.M[u,v]$. Thus, $m_i.seqno \leq m_j.M[u,v]$.

Thus, by induction, $m_i.snd \rightarrow_h m_j.snd \;\Rightarrow\; m_i.seqno \leq m_j.M[u,v]$.

(A.2) $\langle \exists\, m_k : \hat{m}_i.dst = \hat{m}_k.dst : (\hat{m}_i.snd \prec_s \hat{m}_k.snd) \wedge (m_k.snd \rightarrow_h m_j.snd) \rangle \;\Rightarrow$
$m_i.seqno \leq m_j.M[u,v]$
Since $\hat{m}_i.dst = \hat{m}_k.dst$ and $\hat{m}_i.snd \prec_s \hat{m}_k.snd$ therefore $m_i.seqno < m_k.seqno$. Moreover, since $m_k.snd \rightarrow_h m_j.snd$, using (A.1) we have $m_k.seqno \leq m_j.M[u,v]$. Combining both the results, we have $m_i.seqno \leq m_j.M[u,v]$.

($\Leftarrow$)
Assume $m_i.seqno \leq m_j.M[u,v]$. Using Lemma 14, we can infer that there exists a message m_l such that $m_l.seqno = m_j.M[u,v]$ and $m_l.snd \rightarrow_h m_j.snd$. Moreover, $\hat{m}_l.src = S_u, \hat{m}_l.dst = S_v$. Since

$\hat{m}_i.src = S_u = \hat{m}_l.src$, $\hat{m}_i.dst = S_v = \hat{m}_l.dst$ and $m_i.seqno \leq m_j.M[u,v] = m_l.seqno$, therefore $\hat{m}_i.snd \preccurlyeq_s \hat{m}_l.snd$. ■

Theorem 16 *The static module implements CO'' under the assumption that the channels among mobile support stations are FIFO.*

Proof: Let $\mathcal{X}_{SM}$ and $\mathcal{X}_{CO''}$ be the set of executions accepted by the proposed static module and the condition CO'' respectively. To prove that the static module implements CO'', we need to show that $\mathcal{X}_{SM} = \mathcal{X}_{CO''}$ i.e. the executions generated by the static module satisfy the condition CO'' and vice versa. For convenience, let $m_i \mapsto m_j \stackrel{def}{=} \langle \exists\, m_k : \hat{m}_i.dst = \hat{m}_k.dst : (\hat{m}_i.snd \preccurlyeq_s \hat{m}_k.snd) \wedge (m_k.snd \rightarrow_h m_j.snd) \rangle$. Observe that $m_i \mapsto m_j \Rightarrow m_i \rightarrow_s m_j$. Therefore $\mapsto$ is acyclic.

(B.1) $\mathcal{X}_{CO''} \subseteq \mathcal{X}_{SM}$: Consider an execution $\mathcal{X}$ that satisfies CO''. Let $\rightarrow$ denote the Lamport's "happened before" relation on the set of events (on MHs and MSSs) in the execution $\mathcal{X}$. Since $\rightarrow$ is a partial order, it can be extended to some total order. Let E denote the sequence of events with respect to the total order and E_n be the prefix of E containing the first n events. We prove that for all n, E_n can be generated by the proposed static module. The proof is by induction on n. For the purpose of the proof, the events are either *deliver* or *non-deliver* events. Note that the static module controls only the deliver events on mobile support stations.

Base Case ($n = 1$): Observe that the first event cannot be a deliver event. Therefore E_1 can be generated by the static module.

Induction Step ($n > 1$): Using induction hypothesis, E_{n-1} can be generated by the static module. Assume n^{th} event, say e_n, is a deliver event on a mobile support station, say S_v, and let m_i be the application message involved in the event. We need to prove that m_i is deliverable according to our static module. Let $\mathcal{M}_R$ denote the set of messages destined for $m_i.dst$ that have been received but not yet delivered at S_v just before e_n occurs ($\mathcal{M}_R \neq \phi$ since $m_i \in \mathcal{M}_R$). Let $chann(G, e_n)$ denote the set of messages sent to S_v in-transit (sent to S_v but not yet received at S_v) in the consistent cut G that includes e_n, and $\mathcal{M}_D$ be $\mathcal{M}_R \cup chann(G, e_n)$. We first show that m_i is minimal in $\mathcal{M}_D$ with respect to $\mapsto$ ($\mapsto$ is acyclic). Assume the contrary. Let m_k be the application message such that $m_k \mapsto m_i$. Then $m_k \in \mathcal{M}_R$ or $m_k \in chann(G, e_n)$. In either case, $\mathcal{X}$ does not satisfy CO'', a contradiction. Now we prove that m_i is deliverable according to the proposed static module. We prove the contrapositive, that is, if m_i is not deliverable then it is not minimal in $\mathcal{M}_D$. From the static module, it can be verified that either (1) $lastrcvd_v[u] < m_i.M[u,v]$ for some S_u, or (2) there exists an application message m_k in $rcvQ_v$, destined for $m_i.dst$, such that $m_k.seqno \leq m_i.M[u,v]$, where $\hat{m}_k.src = S_u$. In the first case, (1), using Lemma 14 we can infer that there exists a message m_k such that $\hat{m}_k.src = S_u$ and $\hat{m}_k.dst = S_v$. Also, $m_k.seqno \leq m_i.M[u,v]$ and $m_k \in chann(G, e_n)$. Using Lemma 15, we have $m_k \mapsto m_i$. In the second case, (2), $m_k \in \mathcal{M}_R$. Again using Lemma 15, we can conclude that $m_k \mapsto m_i$. In either case m_i is not minimal in $\mathcal{M}_D$, a contradiction. Thus, m_i is deliverable according to the static module.

Therefore, using induction, we can infer that the execution $\mathcal{X}$ can be generated by the static module.

(B.2) $\mathcal{X}_{SM} \subseteq \mathcal{X}_{CO''}$: Consider an execution $\mathcal{X}$ generated by the static module. We have to prove that $\mathcal{X}$ satisfies CO''. Let m_i and m_j be arbitrary application messages such that $m_i \mapsto m_j$. If $\hat{m}_i$ and $\hat{m}_j$ are destined for different MSSs then CO'' is trivially satisfied. Hence assume $\hat{m}_i.dst = \hat{m}_j.dst$. Let

$\hat{m}_i.src = S_u$ and $\hat{m}_i.dst = \hat{m}_j.dst = S_v$. Using Lemma 15, we can conclude that $m_i.seqno \leq m_j.M[u,v]$. From the protocol it can be verified that when $\hat{m}_j.dlv$ occurs then $lastrcvd_v[u] \geq m_j.M[u,v]$. Therefore $lastrcvd_v[u] \geq m_i.seqno$ i.e. $\hat{m}_i.rcv$ has already occurred. Thus, we have $\neg(\hat{m}_j.dlv \prec_s \hat{m}_i.rcv)$. If m_i and m_j are destined for different MHs then the first expression in the consequent of CO'' trivially holds. Therefore assume $m_i.dst = m_j.dst = h_d$. Again from the protocol it can be verified that when $\hat{m}_j.dlv$ occurs then $\hat{m}_i$ is not in $rcvQ_v$. Since $\hat{m}_i$ has been received (as argued before) therefore $\hat{m}_i.dlv$ has already occured at S_v (when $\hat{m}_j.dlv$ occurs). Moreover, the wireless channels are FIFO and reliable. Thus, we have $\neg(m_j.dlv \prec_s m_i.dlv)$. Hence $\mathcal{X}$ satisfies CO''.

Thus, $\mathcal{X}_{SM} = \mathcal{X}_{CO''}$ and the theorem holds. ■

Although we do not prove here but if we relax the FIFO assumption then it can be easily verified that the static module Section 4.1 implements $CO'' \wedge FO''$.

5 Comparison and Discussion

The proposed static module implements $CO'' \wedge FO''$ which is weaker than CO' implemented by $AV2$ ($CO' \Rightarrow CO'' \wedge FO''$). As a result, unnecessary delay in our protocol is lower than that imposed in $AV2$. In the worst case, message overhead in our protocol is $O(n_s^2 + n_h)$ but we expect it to be closer to $O(n_s^2)$ in practice. Our storage overhead in each MSS is $O(k \times n_s^2)$, where k is the number of MHs currently in the cell of the MSS. Even though this overhead is higher than that of $AV2$, it can be easily accommodated by MSSs due to their rich memory resources.

PSR [17] is not suitable for systems where the number of mobile hosts dynamically changes because the structure of information carried by each message in their algorithm depends on the number of participating processes. In our protocol, the structure of the information carried by each message in the wired network does not vary with the number of MHs in the system. So, our protocol is more suitable for dynamic systems. PSR, however, incurs no unnecessary delay in message delivery.

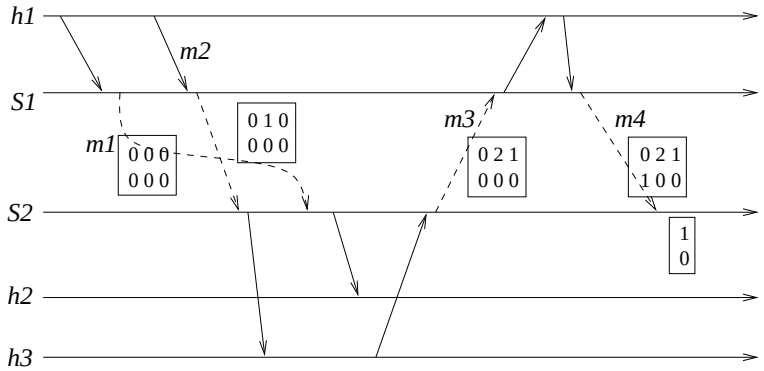


Figure 10: A mobile computation illustrating the liveness problem in YHH.

We first give a scenario (in Figure 10) where YHH does not satisfy liveness property. According to YHH, message m_4 will be delayed because $m_4.M[1,2] > \text{MH_DELIV}_2[1]$. And since at the time when m_4 arrives at S_2, there are no messages in transit, m_4 is delayed indefinitely. The problem can be corrected by using sequence numbers. The static module in YHH (corrected) [22] satisfies $\hat{m}_i.snd \rightarrow_s \hat{m}_j.snd \Rightarrow \neg(m_j.dlv \prec_h m_i.dlv)$. Their message overhead in the wired network is $O(n_s \times n_h)$. This overhead is higher than ours but lower than $AV1$. Their unnecessary delay is strictly lower than $AV2$. When comparing in

terms of unnecessary delay, their delay is lower than ours in the average case which is expected because of their higher message overhead. However, there are cases where our protocol does not impose delivery delay but their protocol does. One can further reduce the unnecessary delay in *YHH* using the technique introduced in this paper. By assigning a matrix of size $n_s \times n_h$ to each host, the condition implemented by their static module can be weakened to,

$$\langle \exists\, m_k : m_i.dst = m_k.dst : (\hat{m}_i.snd \preccurlyeq_s \hat{m}_k.snd) \wedge (m_k.snd \rightarrow_h m_j.snd) \rangle \;\Rightarrow\; \neg(m_j.dlv \prec_h m_i.dlv)$$

Table 1 summarizes the comparison between our protocol and the previous work.

Algorithm	Message overhead	Well-suited for dynamic systems
$AV2$	$O(n_s^2)$	Yes
PSR	$O(n_h^2)$	No
YHH	$O(n_s \times n_h)$	No
Our Algorithm	$O(n_s^2 + n_h)$	Yes
n_h: the number of mobile hosts n_s: the number of mobile support stations		

Table 1: Comparison between our algorithm and the previous work.

6 Performance Evaluation

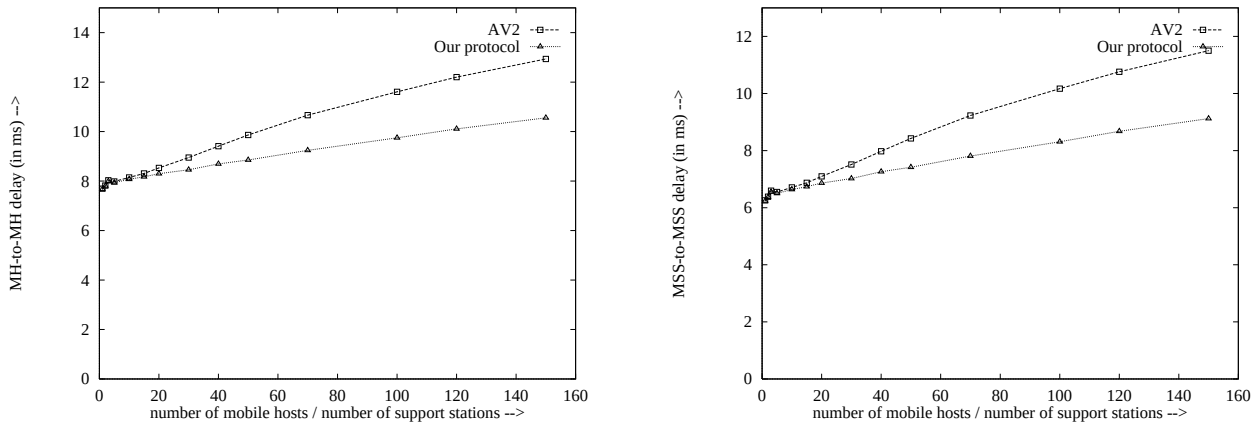
6.1 Simulation Environment

Simulation experiments are conducted for different combinations of *message size* and *communication patterns*. We use 512 bytes for the size of small messages, and $8K - 10K$ bytes for large messages. Two communication patterns are used in the simulation: *uniform*, and *nonuniform*. Nonuniform pattern is induced by having odd numbered hosts generate messages at three times the rate of even numbered hosts. For each application message m, we define *MH-to-MH Delay* as the elapsed time between $m.snd$ and $m.dlv$. Similarly, *MSS-to-MSS Delay* is the elapsed time between $\hat{m}.snd$ and $\hat{m}.dlv$.

The time between generation of successive messages at a mobile host is exponentially distributed with mean 100 ms. The destination host of each message is a uniformly distributed random variable. The throughput of a wired channel is assumed to be 100 Mbps, and the propagation delay in a wired channel is 7 ms. These two parameters are also used in [5]. For a wireless channel, the throughput and propagation delay are respectively assumed to be 20 Mbps and 0.5 ms. This throughput of wireless links is supported in European High Performance Radio Local Area Network (HiperLAN). In each run, the ratio of the number of mobile hosts and support stations is varied from 1 to 150.

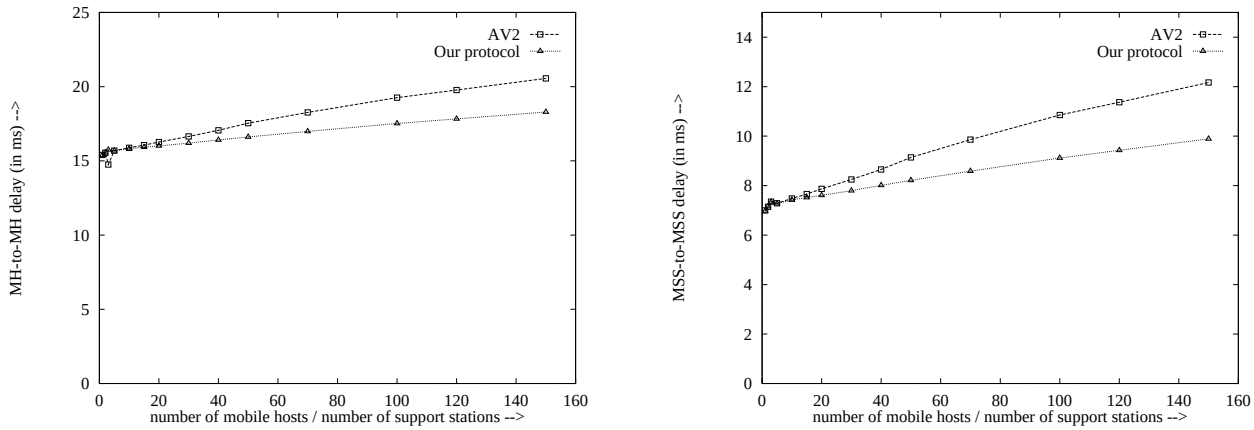
6.2 Results

We plot the MH-to-MH and MSS-to-MSS delay from our static module against those from $AV2$.



(a) (b)

Figure 11: Delay under uniform communication pattern and small message size.



(a) (b)

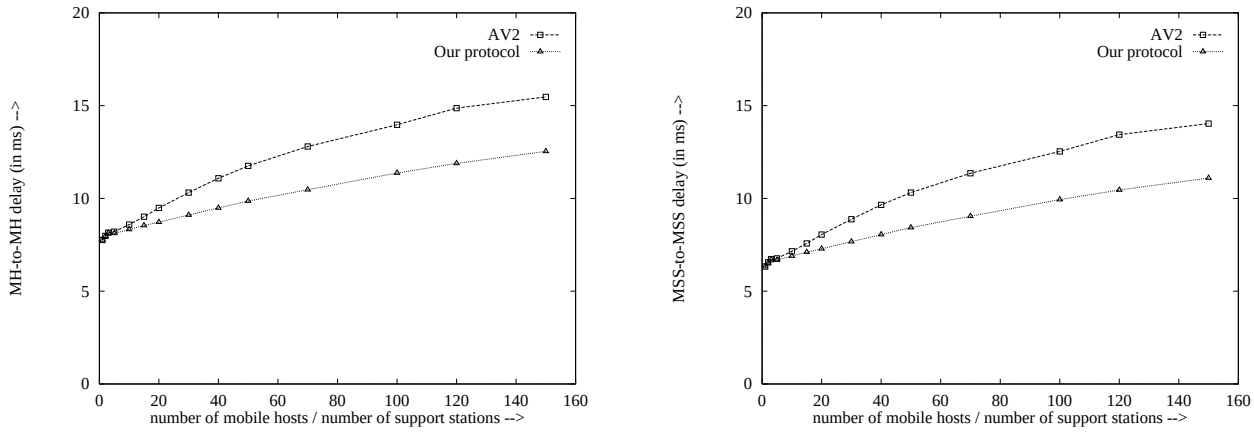
Figure 12: Delay under uniform communication pattern and large message size.

Figure 11(a) and Figure 11(b) present MH-to-MH and MSS-to-MSS delays respectively under uniform communication pattern and small message size. The result shows that our static module can reduce the MH-to-MH delay by as much as 18.4%, and 20.7% for MSS-to-MSS delay.

Figure 12(a) and Figure 12(b) present MH-to-MH and MSS-to-MSS delays respectively under uniform communication pattern and large message size. The result shows that our static module can reduce the MH-to-MH delay by as much as 11.02%, and 18.7% for MSS-to-MSS delay.

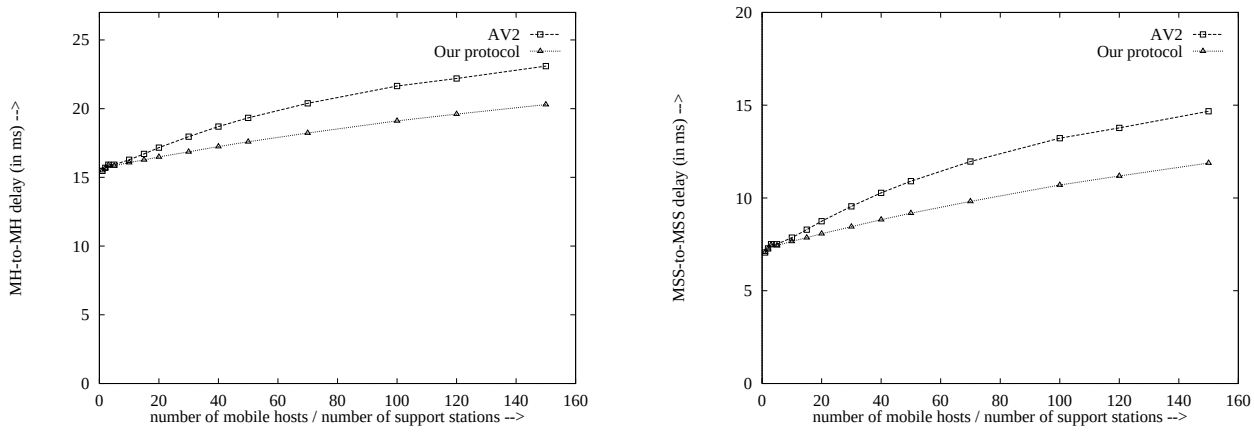
Figure 13(a) and Figure 13(b) present MH-to-MH and MSS-to-MSS delays respectively under nonuniform communication pattern and small message size. The result shows that our static module can reduce the MH-to-MH delay by as much as 18.9%, and 20.9% for MSS-to-MSS delay.

Figure 14(a) and Figure 14(b) present MH-to-MH and MSS-to-MSS delays respectively under nonuniform



(a) (b)

Figure 13: Delay under nonuniform communication pattern and small message size.



(a) (b)

Figure 14: Delay under nonuniform communication pattern and large message size.

communication pattern and large message size. The result shows that our static module can reduce the MH-to-MH delay by as much as 12.11%, and 19% for MSS-to-MSS delay.

7 Conclusion

We have presented an efficient protocol for causal message ordering. This protocal maintains the low message overhead while reducing unnecessary delivery delay imposed by Alagar and Venkatesan's algorithm ($AV2$). Unlike Prakash's and Yen's algorithms, our proposed algorithm is scalable and suitable for dynamic systems because it is easy to adapt to the dynamic changes in the number of mobile hosts. Unlike $AV2$, our handoff module does not require causal ordering among application messages and messages sent as

part of the protocol. This will further reduce the unnecessary delay in our protocol compared to $AV2$. In addition to correctness proofs for static and handoff modules, we also present the condition implemented by our static module. The conditions implemented by $AV2$ and Yen's static modules are also provided. Simulation results show that for small messages, our protocol can significantly reduce the end-to-end delay. Finally, we provide a case where Yen's algorithm does not satisfy liveness property, that is, it is possible that a message is delayed indefinitely.

References

[1] Arup Acharya and B. R. Badrinath. Delivering multicast messages in networks with mobile hosts. In *Proceedings of the 13th International Conference on Distributed Computing Systems*, pages 292–299, May 1993.

[2] F. Adelstein and M. Singhal. Real-time Causal Message Ordering in Multimedia Systems. In *Proceedings of 15th Internationcal Conference on Distributed Computing Systems*, pages 36–43, June 1995.

[3] M. Ahamad, P. Hutto, and R. John. Implementing and Programming Ccausal Distributed Memory. In *Proceedings of the 11th IEEE International Conference on Distributed Computing Systems*, pages 271–281, 1991.

[4] M. Ahuja. An Implementation of F-channels. *IEEE Transactions on Parallel and Distributed Systems*, 4(6):658–667, June 1993.

[5] S. Alagar and S. Venkatesan. Causal Ordering in Distributed Mobile Systems. *IEEE Transactions of Computers*, 6(3), March 1997.

[6] O. Babaoglu and K. Marzullo. Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanisms. In Sape Mullender, editor, *Distributed Systems*, pages 55–96. Addison-Wesley, 1993.

[7] P. Bhagwat and C.E. Perkins. A Mobile Networking System Based on Internet Protocol (IP). In *Proceedings of the USENIX Symposium on Mobile and Location-Independent Computing*, pages 69–82, August 1993.

[8] K. Birman and T. Joseph. Reliable Communication in Presence of Failures. *ACM Transactions on Computer Systems*, 5(1):47–76, February 1987.

[9] K. Birman, A. Schiper, and P. Stepehenson. Lightweight causal and atomic broadcast. *ACM Transactions on Computer Systems*, 9(3):272–314, 1991.

[10] B. Charron-Bost, F. Mattern, and G. Tel. Synchronous and Asynchronous Communication in Distributed Computations. Technical Report TR91.55, LITP, University Paris 7, France, September 1991.

[11] G. Cho and L.F. Marshall. An Efficient Location and Routing Scheme for Mobile Computing Environments. *IEEE Journal on Selected Areas in Communications*, 13(5):868–879, June 1995.

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

[13] David B. Johnson. Scalable and Robust Internetwork Routing for Mobile Hosts. In *Proceedings of the 14th International Conference on Distributed Computing Systems*, pages 2–11, June 1994.

[14] A.D. Kshemkalyani and M. Singhal. An Optimal Algorithm for Generalized Causal Message Ordering. In *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing*, pages 87–88, Philadephia, Pennsylvania, May 1996.

[15] L. Lamport. Time, Clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[16] A. Mostefaoui and M. Raynal. Causal Multicasts in Overlapping Groups: Towards a Low Cost Approach. In *Proceedings of the 4th IEEE International Conference on Future Trends in Distributed Computing Systems*, pages 136–142, Lisbon, September 1993.

[17] R. Prakash, M. Raynal, and M. Singhal. An efficient causal ordering algorithm for mobile computing environments. In *Proceedings of the 16th International Conference on Distributed Computing Systems*, 1996.

[18] M. Raynal, A. Schiper, and S. Toueg. Causal Ordering abstraction and a simple way to implement it. In *Information Processing Letters*, volume 39(6), pages 343–350, 1991.

[19] M. Raynal, G. Thia-Kime, and M. Ahamad. From Serializable to Causal Transactions for Collaborative Applications. Technical Report 983, Irisa-Rennes, France, February 1996. 22 pages.

[20] L. Rodrigues and P. Verissimo. Causal Separators for Large-Scale Multicast Communication. In *Proceedings of the 15th IEEE International Conference on Distributed Computing Systems*, pages 83–91, Vancouver, June 1995.

[21] A. Schiper, J. Eggli, and A. Sandoz. A New Algorithm to Implement Causal Ordering. In *Proceedings of the 3rd International Workshop on Distributed Algorithms*, LNCS-392, pages 219–232, Berlin, 1989.

[22] Li-Hsing Yen, Ting-Lu Huang, and Shu-Yuen Hwang. A Protocol for Causally Ordered Message Delivery in Mobile Computing Systems. *Mobile Networks and Applications*, 1997.