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
Electrical and Computer Engineering Department
The University of Texas at Austin,
Austin, Texas 78712

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.

A mobile computing system consists of two kinds of processing units: *mobile hosts*, and *mobile support stations*. A mobile host (MH) is a host that can move while retaining its network connections. A mobile support stations (MSS) is a machine that can communicate directly with mobile hosts. The coverage area under an MSS is called a *cell*. Even though cells may physically overlap, an MH can be 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. Throughout the paper, we use the terms mobile host and host, and mobile support station and support station interchangeably.

In this paper, we consider causal message ordering required in many distributed applications such as management of replicated data [10, 11], distributed monitoring [8], resource allocation [19], distributed shared memory [4], and multimedia systems [1]. Algorithms to implement causal

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

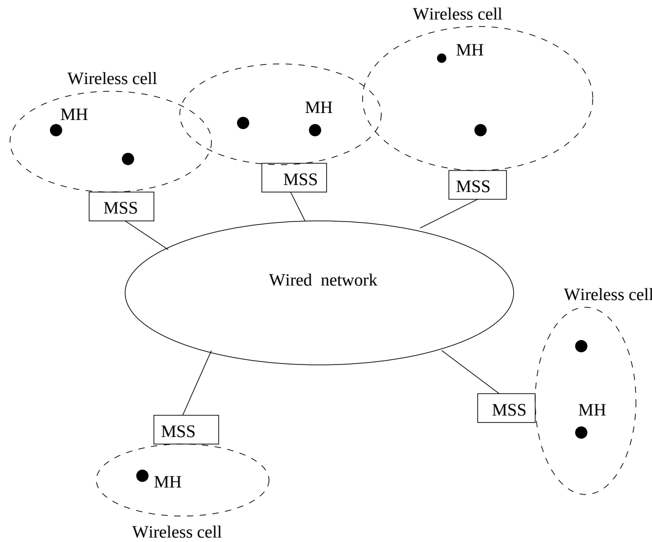

Figure 1: A mobile computing system.

message ordering in systems with static hosts have been presented in [3, 11, 17, 19, 20, 21]. These algorithms, however, require high message and memory overheads; therefore, they cannot be directly employed in mobile computing systems. We propose a new protocol suited to mobile computing systems in which message overhead is small compared 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 more scalable than existing protocols since our message overhead is independent of the number of hosts in the system.

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 [6] proposed three algorithms based on the algorithm presented in [19]. The first algorithm($\mathcal{AL}_1$) maintains causal ordering among all MHs. Message overhead is therefore proportional to the square of the number of hosts(n_h). However, data structures required in the algorithm are stored in MSSs to reduce load on mobile hosts and wireless links. In the second algorithm($\mathcal{AL}_2$), causal ordering is exclusively maintained among MSSs. This is sufficient for causal ordering among MHs only when each wireless channel is FIFO and MHs never change cell. Message overhead reduces to the square of the number of MSSs(n_s). However, the procedure for handling host migration (*handoff*) is more complex than that of the first algorithm. Since stronger ordering is imposed, messages may experience unnecessarily delay even though they do not violate causal ordering among mobile hosts. Their third algorithm($\mathcal{AL}_3$) is aimed to reduce this unnecessary delay by partitioning each physical MSS into k logical support stations. As k increases, the degree of unnecessary delay decreases, but message overhead and the cost of handling host migration increase.

Yen, Huang, and Hwang [22] proposed another algorithm based on [19]. Message overhead in their algorithm falls between $\mathcal{AL}_1$ and $\mathcal{AL}_2$. In particular, each MSS maintains a matrix of size $n_s \times n_h$; this matrix is attached to each message sent by an MSS. Unnecessary delay in this algorithm is lower than $\mathcal{AL}_2$. Handoff protocol in this algorithm is also less complicated than $\mathcal{AL}_2$.

Prakash, Raynal, and Singhal [18] 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 $O(n_h^2)$. Furthermore, the size of their message overhead varies when the number of participating processes dynamically changes. This make their algorithm not suitable for dynamic systems.

In the proposed protocol, we are able to decrease the unnecessary delivery delay while maintaining message overhead at $O(n_s^2 + n_h)$, in the worst case. Our handoff protocol is more efficient than that in $\mathcal{AL}_2$ and $\mathcal{AL}_3$ because we do not require causal ordering among messages sent as part of the handoff. We also provide the *formal* proof for both static and handoff protocols. Furthermore, the condition for which messages are delayed in the protocol is also formally stated and proved.

2 System model

A message passing mobile computation consists of a set of n_h processes running on mobile hosts, $\mathcal{H} = \{h_i \mid 1 \leq i \leq n_h\}$. Let $\mathcal{S}$ be the set of mobile support stations, $S_1, \ldots, S_{n_s}$. We use $\mathcal{H}_i$ to denote the set of mobile hosts in the cell of S_i. In general, $n_h \gg n_s$. These MH processes do not share a global memory or a global clock, and they communicate asynchronously with each other. Each process 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 variables and the program counter in the process. Events in each process are classified into three types: send events, receive events, and local events. Delivery events are local events that represent the delivery of a received message to the application or applications running in that process.

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 sent between a MH process and a MSS process. A dashed arrow represents a message sent from a 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*.

For any two events, e and f on some mobile host, we write $e \prec_h f$ iff e occurs before f. We use $\rightarrow_h$ to denote the Lamport's *happened before* relation [16] in the abstract diagram. Similarly, $e \prec_s f$ iff e occurs before f on some mobile support station. Also, let $\rightarrow_s$ denote the Lamport's happened before relation in the concrete diagram.

A *data message* is a message sent by an MH intended for another MH. Since mobile hosts do not communicate with each other directly, an MH, say h_s, send a data message m to its local support station, say S_i, which then forwards it to the local support station, S_j, of the destination host, h_d. Using our notation, $m.src$ and $m.dst$ denote the source and the destination hosts 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 data message m along with a matrix for ensuring causality), requesting it to deliver m to h_d. Again, $\hat{m}.src$ denotes the support station of h_s (in this case S_i) when m is dispatched. Similarly, $\hat{m}.dst$ denotes the support station

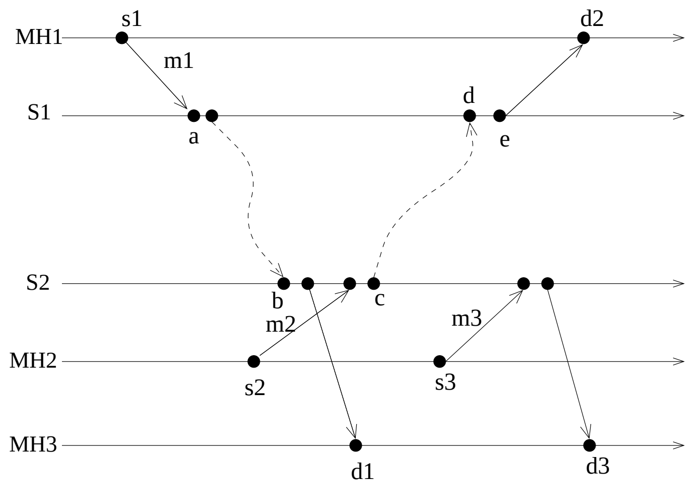

Figure 2: A concrete diagram of a mobile computation

to which S_i forwards m (in this case S_j). As before using our notation, $\hat{m}.snd$ denotes the send event of $\hat{m}$ on the support station S_i. Similarly, $\hat{m}.rcv$ and $\hat{m}.dlv$ denote the receive and delivery events respectively of $\hat{m}$ on S_j. When it is clear from the context, we use *message* in place of *data message*.

For any two messages m_1 and m_2, we say that m_1 causally precedes m_2 in abstract view if $m_1.snd \rightarrow_h m_2.snd$. We say that m_1 causally precedes m_2 in concrete view if $\hat{m}_1.snd \rightarrow_s \hat{s}_2.snd$ We assume that every message sent in both wired and wireless networks is eventually received, and there are no spurious messages. We also assume that messages exchanged between any two MSSs are received in the order sent, and all wireless channels are FIFO.

3 Sufficient Conditions

A mobile computation is causally ordered if the following property holds for any two messages, m_1 and m_2

$$m_1.snd \rightarrow_h m_2.snd \quad \Longrightarrow \quad \neg(m_2.dlv \prec_h m_1.dlv) \qquad (\mathcal{CO})$$

We next show the sufficient conditions for causal message ordering in mobile computation.

Theorem 1 *: A mobile computation with multiple MSSs is causally ordered if*

(C_1) all wireless channels are FIFO,
(C_2) messages in the wired network is causally ordered, and
(C_3) each MSS sends out messages in the order they are received.

Proof: Let message m_1 be sent from h_i to h_j and message m_2 be sent from h_k to h_j. Given $m_1.snd \rightarrow_h m_2.snd$, we need to show that if C_1, C_2, and C_3 are satisfied, then m_1 and m_2 are delivered at h_j in that order.

Since there is no direct communication between MHs, each message from an MH to another MH must be sent through the MSS(s). From $m_1.snd \rightarrow_h m_2.snd$, there must be a message path from h_i to h_k via S_i and S_k if S_i is the MSS of h_i, and S_k is the MSS of h_k. From C_1 and the fact that $m_1.snd \rightarrow_h m_2.snd$, $\hat{m_1}.snd \rightarrow_s \hat{m_2}.snd$. Note that this is still true even if h_i and h_k are in the same cell, or there are more than one message involved in the causal chain between s_1 and s_2. From C_2, it implies that m_1 will be delivered by S_j before m_2. By C_1 and C_3, h_j will deliver m_1 and m_2 in that order. Figure 3 illustrates a causally ordered computation in which all MHs are located in different MSSs. ∎

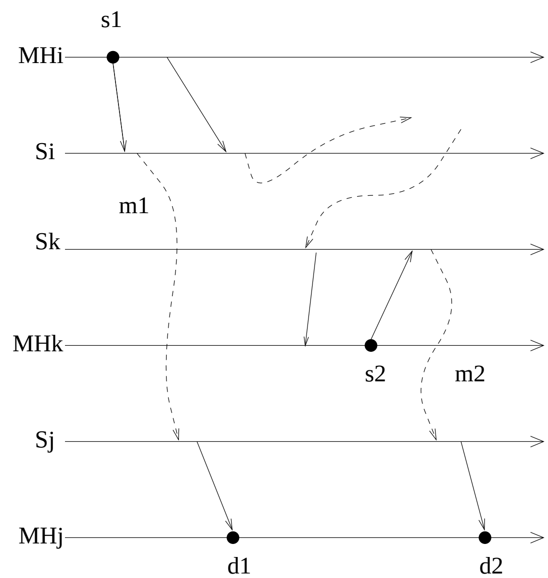

Figure 3: A concrete diagram showing a causally ordered mobile computation.

Sufficient conditions shown in Theorem 1 were implicitly used in [6]. 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 one cell, causal ordering can be provided by simply using FIFO between MHs and the MSS. This is significantly more efficient than using matrices as in [19] although it is centralized.

We show that C_1, C_2, and C_3 are not necessary by a counter-example. In Figure 4, $s_1 \rightarrow s_3$ and $d_1 \prec d_3$. Therefore, this mobile computation is causally ordered, but C_1 and C_2 do not hold.

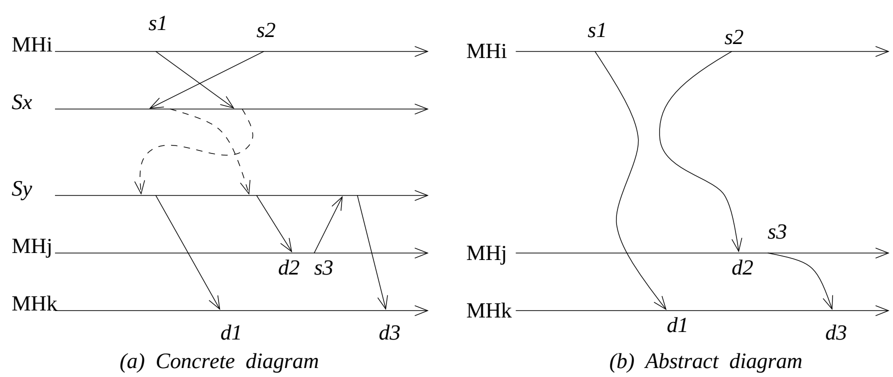

Figure 4: A counter-example to show that C_1, C_2, and C_3 are not necessary.

Let us consider a computation in Figure 5. In this example, MH_a is in the cell of S_i, MH_b and

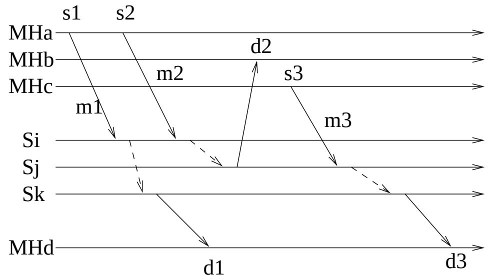

Figure 5: A mobile computation in which $m_1 \rightarrow_s m_3$, but $m_1 \not\rightarrow m_3$.

MH_c in the cell of S_j, and h_d in the cell of S_k. Since there does not exist a message path from s_1 to s_3 in the abstract diagram of the computation in Figure 5, $m_1 \not\rightarrow_h m_3$. Therefore, m_3 can be delivered to h_d before m_1 without violating CO. However, $m_1 \rightarrow_s m_3$. Observe that if $m \rightarrow_h m'$, then $m \rightarrow_s m'$ when the channel between each MH and its MSS is FIFO.

We can formally state condition C_2 as follows:

$$\hat{m}_1.snd \rightarrow_s \hat{m}_2.snd \quad \Longrightarrow \quad \neg(\hat{m}_2.dlv \prec_s \hat{m}_1.dlv) \qquad (\mathcal{CO}')$$

The algorithm presented by Alagar and Venkatesan [6] enforces $\mathcal{CO}'$ in order to achieve $\mathcal{CO}$. This algorithm delays messages that violate $\mathcal{CO}'$ even though they do not violate $\mathcal{CO}$. This can be illustrated in a computation in Figure 6. In this example, message m_1 does not causally precede m_3 in the abstract view, but it does in the concrete view. With $\mathcal{CO}'$, m_3 is unnecessarily delayed until m_1 is delivered.

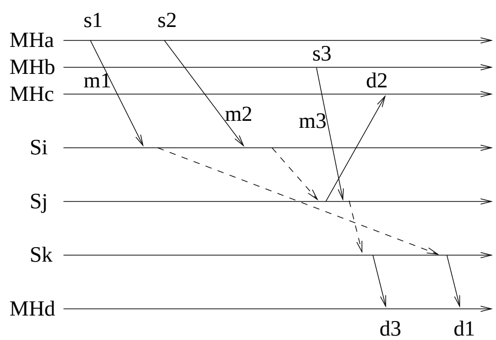

Figure 6: An example of our implementation

Our goal is to reduce this extra delay, while maintaining the size of message overhead in the wired network at $O(n_s^2)$.

4 The Protocol

To reduce the unnecessary delay in $\mathcal{AL}_2$, we propose a new protocol that implements a property weaker than $\mathcal{CO}'$. We first introduce the protocol, and then formally state the implemented property.

Static protocol

Our static protocol is based on the algorithm proposed by Raynal et al. [19]. We assign the following data structures to each mobile host h_p: (1) an integer matrix, M_p, of size $n_s \times n_s$, (2) a message queue, $ackQ_p$. Both are maintained by the local MSS of h_p. Each support station, S_i, also maintains the following data structures for itself: (1) a message queue, $rcvQ_i$, and (2) two integer arrays, $lastrcvd_i$ and $lastsent_i$, of size n_s. The static algorithm is given in Figure 7. For the simple exposition of the protocol, we here assume that channels among MSSs are FIFO.

Whenever an h_p wants to send a message m to h_q, h_p must first send m to its local support station, say S_i. Then, S_i increments $lastsent_i[j]$ (let S_j be the local MSS of h_q), and attaches $lastsent_i[j]$, and matrix M_p to m before sending m to S_j. S_i then updates entry $M_p[i,j]$ by $lastsent_i[j]$.

Once message $m(M_u, seqno)$ arrives at S_j, it is added into $rcvQ_i$. Note that channels in the wired network are assumed FIFO. At this point, we say that m is *received* at S_j. A received message is deliverable to h_q when conditions in step (A4) are satisfied. The delivered message is removed from $rcvQ_i$ and added into the $ackQ_p$. Messages stored in $ackQ_p$ are sent, in sequence, to h_p over a wireless link. S_j waits for an acknowledgement from h_d, before matrix M_p is updated according to (A6). This prevents m from being considered causally preceded any outgoing message from h_d that is sent before m is received by h_d.

In the following section, we prove that the static protocol implements $\mathcal{CO}''$ under assumption that channels among MSSs are FIFO. We can formally state $\mathcal{CO}''$ as follows. For any message m_1 and m_2,

$$\begin{aligned} &\langle \exists m_k : \hat{m}_i.dst = \hat{m}_k.dst : (\hat{m}_i.snd \preceq \hat{m}_k.snd) \wedge (m_k.snd \rightarrow_h m_j.snd)\rangle \\ &\qquad\qquad \Longrightarrow \qquad \neg(m_j.dlv \prec_h m_i.dlv) \wedge \neg(\hat{m}_j.dlv \prec_s \hat{m}_i.rcv) \end{aligned} \qquad (\mathcal{CO}'')$$

where $e \preceq f$ iff $(e = f) \vee (e \prec_s f)$.

For a fair comparison with the previous protocols, we have to state the property implemented by our protocol without the assumption that channels among MSSs are FIFO. If the channels among support station are not FIFO then the static protocol satisfies,

$$\mathcal{CO}'' \ \wedge\ \hat{m}_1.snd\ \prec_s\ \hat{m}_2.snd \implies \neg(\hat{m}_2.dlv\ \prec_s\ \hat{m}_1.rcv)$$

Handoff protocol

To ensure causal ordering ($\mathcal{CO}$) when MHs move, handoff protocol must be executed every time an MH changes cell. This can be illustrated by the following example. Let $m_1 \rightarrow_h m_2$ and both are intended for the same MH h_d. Assume that h_d moves from the cell of S_i to S_j, it leaves S_i before m_1 arrives. Also, assume that m_2 is sent to S_j. It is easy to see that with only static protocol, m_2 will be delivered to h_d before m_1 violating $\mathcal{CO}$.

Our handoff protocol is more efficient than Alagar's handoff protocol. This is because we do not require causal ordering among data messages and messages sent as part of the handoff protocol. The handoff protocol is given in Figure 9 and Figure 10. The modification of the static protocol due to host mobility is shown in Figure 8

In our handoff protocol, each MH h maintains an integer, *mbl* (mobility count), initially 0. *mbl* is incremented each time mobile host switches cell. Each support station S_i maintains an array $cell_i[1 \ldots n_h]$ of pair $\langle mbl, mss\rangle$. Entry $cell_i[d].mss$ represents the current location(cell) of host d

known by S_i. $cell_i[d].mbl$ is the mobility number associated with $cell_i[d].mss$. We assume that initially each MSS knows the exact location of each MH.

Here we give a brief description of our handoff protocol. We refer to messages sent as part of the handoff protocol as *signals*. Consider a scenario when a mobile host h moves from S_i to S_j. Once h enters the cell of S_j, it sends a signal $register(mbl, S_i)$ to S_j to inform S_j of its presence. On receiving $register$ from h, S_j updates $cell[h]$ with $\langle mbl, S_i\rangle$, and sends $handoff\_begin(h, mbl)$ signal to S_i.

When S_i receives $handoff\_begin(h, mbl)$, S_i updates its own $cell[h]$, and sends *enable* signal along with M_h and $ackQ_h$ to S_j. S_i then broadcasts $notify(h, mbl, S_j)$ to all MSSs except S_i and S_j. On receiving *enable* from S_i, S_j resends all messages in $ackQ_h$. Then, S_j can start sending messages on behalf of host h. However, messages destined for h must wait until S_j receives *handoff_over* signal from S_i.

When an MSS S_k receives $notify$ from S_i, S_k updates $cell[h]$, and sends $last(h)$ back to S_i. Since messages among MSSs are FIFO, when S_i delivers *last* from S_k, it implies that there is no messages in transition sending from S_k to S_i intended for host h.

When messages intended for h received by S_i after S_i receives *handoff_begin* and before S_i receives all *last* signals become deliverable (step A4 in static protocol), S_i marks them as *old* and forwards to S_j. Once S_i receives *last* from each support station except S_j, it sends *handoff_over* to S_j.

Since messages in the wired network are not causally ordered, for any two messages, m_1 and m_2 intended for the same host d such that $m_1 \rightarrow_h m_2$, it is possible that m_1 is sent to the *new* MSS, but m_2 is sent the *old* MSS. To ensure $\mathcal{CO}$, we attach additional information, *up_cell*, to each data message, data message tagged as *old*, and *enable* signal sent from any MSS S_i to S_j (steps A2', A5', A13). *up_cell* is a list of 3-tuple, $\langle h_k, cell[k].mbl, cell[k].mss\rangle$, for each host h_k that has changed cell according to S_i's knowledge since *up_cell* last sent to S_j.

When S_j receives messages (signals) attached with *up_cell*, S_j updates $cell_j[k]$ if the location of h_k in *up_cell* is more updated than that in $cell_j[k]$, that is, the mobility count in $cell_j[k]$ is less than that in *up_cell* (step A3').

The handoff protocol terminates at S_j after $handoff\_over(k)$ is received by S_j. If S_j receives $handoff\_begin(k, mbl)$ from some other MSS before the current handoff of host k terminates, S_j will respond to the signal only after the handoff completes.

5 Proof of Correctness

We first prove that the combination of static and handoff protocols implement causal message ordering ($\mathcal{CO}$). Then, we show that our static protocol implements $\mathcal{CO}''$.

5.1 Safety and Liveness Proofs

Here we prove that our static and handoff protocols implement $\mathcal{CO}$.

5.2 Static protocol implements $\mathcal{CO}''$

Here we prove that our static protocol implements $\mathcal{CO}''$.

For the following proofs, we need to define an auxiliary variable *pred* for any message m as follows:

$$m.pred[i,j] = (max\ \mu.seqno : \hat{\mu}.src = S_i, \hat{\mu}.dst = S_j \ \wedge\ \mu \rightarrow_h m)$$

Thus, $m.pred[i,j]$ is the message (say μ) with the biggest *seqno* that causally precedes m and $\hat{\mu}.src = S_i$ and $\hat{\mu}.dst = S_j$. If there is no such message, we define $m.pred[i,j] = \perp$, and $\perp.seqno = 0$.

Let us introduce some notations used in the following Lemma. Let (p,n) denote the sequence of states (*interval*) between the $(n-1)^{th}$ and n^{th} external events (send and delivery events) in h_p. When we say a message m^* is happened before an interval (p,n), we mean that (1) $m^*.snd \rightarrow_h e$, where e is the $(n-1)^{th}$ external event of host p, or (2) $m^*.s$ is e. We also use M_p^n to denote the matrix of host h_p corresponding to the interval (p,n). Note that $M_p^n = m.M$ if m is the message sent initiating the interval $(p, n+1)$.

Lemma 2 $m.M[i,j] = m.pred[i,j].seqno$

Proof: Let m is sent from host p. We prove by induction on n, the number of intervals in p.

Base$(n=1)$: Since the initial matrix is $\mathbf{0}$, the message m initiating the interval $(p,2)$ is tagged with zero matrix. The Lemma follows.

Induction$(n>1)$: Assume true for (p,n) and every interval in the past of (p,n). Suppose the event initiating $(p,n+1)$ is the send of m and the value of *seqno* of (p,n) is ls. By induction hypothesis, $m.M[i,j] = m.pred[i,j].seqno$. From the program text, we know that $m.M = M_p^n$ and $m.seqno = ls+1$. If $\hat{m}.src = S_i$ and $\hat{m}.dst = S_j$, we get $M_p^{n+1}[i,j] = ls+1$. Otherwise, we get $M_p^{n+1}[i,j] = M_p^n[i,j]$. The Lemma follows.

Suppose the event initiating $(p,n+1)$ is the receive of W tagged with matrix M_w. By induction hypothesis, we obtain that $M_w[i,j] = W.pred[i,j].seqno$, and $M_p^n[i,j]$ is the *seqno* of the last message happened before (p,n) sent through S_i and S_j. From step (A6) in the program text, we get that $M_p^{n+1}[i,j]$ is also the *seqno* of the last message happened before $(p,n+1)$ sent through S_i and S_j. ∎

Lemma 3 *For any two messages m_1 and m_2 such that $\hat{m}_1.src = S_i$ and $\hat{m}_2.dst = S_j$, the static protocol satisfies*

$$\langle \exists m_k : \hat{m}_1.dst = \hat{m}_k.dst : (\hat{m}_1.snd \preceq \hat{m}_k.snd) \wedge (m_k.snd \rightarrow_h m_2.snd)\rangle \iff m_1.seqno \leq m_2.M[i,j]$$

Proof:

(⇒)

<u>(A.1) $m_1.snd \rightarrow_h m_2.snd \implies m_1.seqno \leq m_2.M[i,j]$</u>

We prove this by induction on n the number of messages in the causal chain among $m_1.snd$ and $m_2.snd$.

Base Case: $n=0$. It implies that $m_1.snd \prec_h m_2.snd$. From the sending routine, $m_1.seqno \leq m_2.M[i,j]$.

Induction: Let e_n be the last message in the causal chain. By induction, we get $m_1.seqno \leq e_n.M[i,j]$. Since e_n must be delivered to $m_2.dst$ before $m_2.snd$, we know that $e_n.M[i,j] \leq m_2.M[i,j]$. Therefore, $m_1.snd \rightarrow_h m_2.snd \implies m_1.seqno \leq m_2.M[i,j]$

(A.2) $\langle \exists m_k : \hat{m}_i.dst = \hat{m}_k.dst : (\hat{m}_i.snd \prec_s \hat{m}_k.snd) \wedge (m_k.snd \rightarrow_h m_j.snd) \rangle \implies m_1.seqno \leq m_2.M[i,j]$

From $\hat{m}_1.dst = \hat{m}_k.dst$ and $\hat{m}_1.snd \prec_s \hat{m}_k.snd$,

$$m_1.seqno < m_k.seqno \tag{1}$$

Since $m_k.snd \rightarrow_h m_2.snd$, we get from (A.1) that

$$m_k.seqno \leq m_2.M[i,j] \tag{2}$$

From (1) and (2), we get $m_1.seqno < m_2.M[i,j]$.

($\Leftarrow$)

Let $m_2.M[i,j] = x$, and m_2 be sent from S_k on behalf of h_p. There are two cases:

(B.1) $k \neq i$

From Lemma 2, there must be a message m such that $m.snd \rightarrow_h m_2.snd \wedge \hat{m}.src = S_i, \hat{m}.dst \in S_j \wedge m.seqno = x$. Since $\hat{m}_1.src = S_i, \hat{m}_1.dst = S_j$, and $m_1.seqno \leq x$, we know that $\hat{m}_1.snd \prec_s \hat{m}.snd$.

(B.2) $k = i$

If m_1 and m_2 are sent from the same mobile host, we get from (A2) that $m_1.snd \rightarrow_h m_2.snd$. If they are sent from different mobile hosts, there must be a message m such that $m.snd \prec_h m_2.snd \wedge m.seqno = x$. ■

Theorem 4 *The static protocol implements*

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

where $e \preceq f$ iff $(e = f) \vee (e \prec_s f)$, under the assumption that the channels among support stations are FIFO.

Proof: Let $\mathcal{X}_P$ and $\mathcal{X}_{\mathcal{CO}''}$ be the set of executions accepted by the proposed protocol and condition $\mathcal{CO}''$, respectively. To prove that the static protocol implements $\mathcal{CO}''$, we need to show that $\mathcal{X}_P = \mathcal{X}_{\mathcal{CO}''}$. In the proof, we write $m_i \hookrightarrow m_j$ iff $\langle \exists m_k : \hat{m}_i.dst = \hat{m}_k.dst : (\hat{m}_i.snd \preceq \hat{m}_k.snd) \wedge (m_k.snd \rightarrow_h m_j.snd) \rangle$

$\underline{\mathcal{X}_{\mathcal{CO}''} \subseteq \mathcal{X}_P}$: Let $chann(G,z)$ be the set of all messages sent to the support station S in the channel in the consistent cut G (which includes state $z \in S$). Let $D = min(chann(G,z))$ represent the set of messages minimal in $chann(G,z)$ with respect to $\hookrightarrow$. Let $m(s,d) \in D$. We show that m is deliverable. We show the contrapositive, that is, if m is not deliverable at z, then $m \notin D$.

The fact that m is not deliverable at z, implies that

(1) $\exists x : lastrec_i[x] < m.M_u[x,i]$, or

(2) $\exists m' \in RCVD_i$ intended for h_d and sent from S_k such that $m'.seqno \leq m.M_u[k,i]$

We show that both (1) and (2) falsify $\mathcal{CO}''$. (1). It implies that $\exists m' : lastrec[x] < m'.seqno \leq m.M_u[x,i] \wedge m' \notin rcvQ_i$. From Lemma 3, we know that $m' \hookrightarrow m$ holds. Contradiction. (2). From Lemma 3, we know that $m' \hookrightarrow m$. Contradiction.

$\underline{\mathcal{X}_P \subseteq \mathcal{X}_{\mathcal{CO}''}}$: Given that $m_1 \hookrightarrow m_2$ and both messages are intended for the same host h_d, we need to show that $(m_1.dlv \prec_h m_2.dlv) \vee (\hat{m}_1.rcv \prec_s \hat{m}_2.dlv)$ never hold in any computation accepted by the protocol.

If $(s_1 \rightarrow_h s_2) \vee (\langle \exists s :: B(s_1, s) \wedge s \rightarrow_h s_2 \rangle)$, we know from Lemma 3 that $m_1.seqno \leq m_2.M[i,j]$. If m_1 has been received by S_j but not delivered to h_d, that is, $m_1 \in rcvQ_j$. By step (A4), h_d must deliver m_1 before m_2. If m_1 has not been received by S_j, then $lastrec_j[i] < m_1.seqno$. Therefore, $lastrec_j[i] < m_2.M[i,j]$. Again, from (A4) m_2 must be delayed until m_1 is received by S_j. ∎

6 Discussion

The proposed static protocol implements $\mathcal{CO}''$. This condition is weaker than $\mathcal{CO}'$ implemented by $\mathcal{AL}_2$. As a result, unnecessary delay in our protocol is lower than that imposed in $\mathcal{AL}_2$. In the worst case, message overhead in our protocol is $O(n_s^2 + n_h)$. Our memory overhead in each MSS is O$(k * n_s^2)$, where k is the number of MHs currently in the cell of the MSS. Even though this overhead is higher than that of $\mathcal{AL}_2$, it can be easily accommodated by MSSs due to their rich memory resources.

Prakash's algorithm [18] 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 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 to dynamic systems. Prakash's protocol, however, incurs no unnecessary delay in message delivery.

Yen's static protocol [22] satisfies

$$\hat{m}_1.snd \rightarrow_s \hat{m}_2.snd \quad \Longrightarrow \quad \neg(m_2.dlv \prec_h m_1.dlv)$$

Their message overhead in the wired network is $O(n_s \times n_h)$. This overhead is higher than ours but lower than $\mathcal{AL}_2$. Their unnecessary delay is strictly lower than Alagar's. When comparing in term of unnecessary delay, their delay is lower than ours in the average case. However, there exists cases where our protocol does not impose delivery delay, but their protocol does. Let consider the example given in Figure 6. If m_2 was sent from different MH in cell S_i, and m_3 was sent from h_c, after h_3 delivers m_2, then Yen's algorithm would delay m_3 until m_1 is delivered. In our protocol, m_3 can be deliverd before m_1.

One can further reduce the unnecessary delay in Yen's protocol using technique introduced in this paper. By assigning a matrix of size $n_s \times n_h$ to each host, the enforced condition becomes

$$\langle \exists m_k : m_i.dst = m_k.dst : (\hat{m}_i.snd \preceq \hat{m}_k.snd) \wedge (m_k.snd \rightarrow_h m_j.snd) \rangle \implies \neg(m_j.dlv \prec_h m_i.dlv) \wedge \neg(\hat{m}_j.dlv \prec_s \hat{m}_i$$

where $e \preceq f$ iff $(e = f) \vee (e \prec_s f)$.

The table below summarizes the comparison between our protocol and previous work.

Algorithm	Message overhead	Extra delay in message delivery	Well-suited for dynamic systems
Alagar Venkatesan	$O(n_s^2)$	High	Yes
Prakash et al	$O(n_h^2)$	None	No
Yen et al	$O(n_s \times n_h)$	UD	No
Skawratananond Mittal and Garg	$O(n_s^2 + n_h)$	UD	Yes

n_h: the number of MHs.

n_s: the number of MSSs.

7 Conclusions

conclusion is here.

References

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

[2] M. Ahuja. An Implementation of F-channels. In *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, NO. 6, June 1993, pp. 658-667.

[3] A. D. Kshemkalyani, M. Singhal. An Optimal Algorithm for Generalized Causal Message Ordering. In *Proceedings of the 15^{th} Annual ACM Symposium on Principles of Distributed Computing*, Philadelphia, Pennsylvania, May 1996, pages 87-88.

[4] M. Ahamad, P. Hutto, R. John. Implementing and Programming Causal Distributed Memory. In *Proceedings of the 11^{th} IEEE International Conference on Distributed Computing Systems*, pages 271-281, 1991.

[5] 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,* May 1993, pp. 292-299.

[6] S. Alagar and S. Venkatesan, Causal Ordering in Distributed Mobile Systems, In *IEEE Transactions of Computers*, Vol. 46, No. 3, March 1997.

[7] B. Awerbuch, D. Peleg, Concurrent online tracking of mobile users, In *Proceedings of ACM SIGCOMM Symposium on Communication, Architectures and Protocols*, September 1991, pp. 221-233.

[8] O. Babaoglu and K. Marzullo. Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanisms. In *Distributed Systems,* Edited by Sape Mullender, pp.55-96 , Addison-Wesley, 1993.

[9] 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.

[10] K. Birman and T. Joseph. Reliable Communication in Presence of Failures. In *ACM Transactions on Computer Systems*, February 1987, 5(1):47-46.

[11] K. Birman, A. Schiper, and P. Stephenson. Lightweight Causal and Atomic Broadcast. In *ACM Transactions on Computer Systems*, 9(3):272-314, 1991.

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

[13] G. Cho and L.F. Marshall, An Efficient Location and Routing Scheme for Mobile Computing Environments, In *IEEE Journal on Selected Areas in Communications*, Vol. 13, No. 5, June 1995, pp. 868-879.

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

[15] 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*, September 1991, pp. 235-245.

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

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

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

[19] M. Raynal, A. Schiper, and S. Toueg. Causal Ordering abstraction and a simple way to implement it. *Information Processing Letters*, 39(6):1991, pp. 343-350.

[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. In *Mobile Networks and Applications*, 2(1997) 365-372.

A Appendix

S_i ::

var

$rcvQ$: queue of messages, initially ϕ;
$cell$: array$[1..n_h]$ of pair $\langle mbl, mss\rangle$, 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 $\mathbf{true}$;
$canDeliver$: set of boolean variables, $(\{canDeliver_k \mid h_k \in \mathcal{H}_i\})$, each initially $\mathbf{true}$;

(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 ($canDeliver_d \wedge \langle \forall k :: lastrcvd[k] \geq M[k,i]\rangle \wedge$
$\langle \nexists\ \langle m', M', seqno'\rangle \in rcvQ :: (S_k$ sent m' for $h_d) \wedge (seqno' \leq M[k,i])\rangle$) 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 7: The static protocol for a mobile support station S_i

S_i ::

(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 $cell[d].mss$;
            $lastsent[j]++$;
            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 $\langle m, M_s, lastsent[j]\rangle$ to S_j;
            $M_s[i,j] := lastsent[j]$;
        endwhile;
    endif;

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

(A5') On calling $deliver(\langle m, M, seqno\rangle)$;
    let m be destined for h_d;
    if ($cell[d].mss = S_i$) then
        put $\langle m, M, seqno\rangle$ in $ackQ_d$;
        send m to h_d;
    else
        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 $cell[d].mss\}$;
        send $\langle m, M, seqno, old, up\_cell\rangle$ to $cell[d].mss$;
    endif;

Figure 8: The modification in static protocol in presence of host movement in mobile support station S_i

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 $\mathbf{true}$;
$handoffQ$: 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 9: The handoff protocol for a mobile support station S_i

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 10: The handoff protocol for a mobile support station S_i (contd.)