A Lightweight Algorithm for Causal Message Ordering in Mobile Computing Systems

Chakarat Skawratananond and Vijay K. Garg

TR TR-PDS-1996-017 October 1996

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 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

Abstract

This paper presents a lightweight approach to implement causal message ordering in mobile computing systems. We exploit the tradeoff between message overhead and delay in message delivery to reduce the scalability problem raised by existing algorithms. The proposed algorithm incurs smaller overhead, at the cost of delaying some messages. The size of message overhead in our algorithm is constant, even when the number of processes varies. Therefore, our algorithm scales well and is more suitable for a system in which the number of participating processes is dynamic.

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.

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

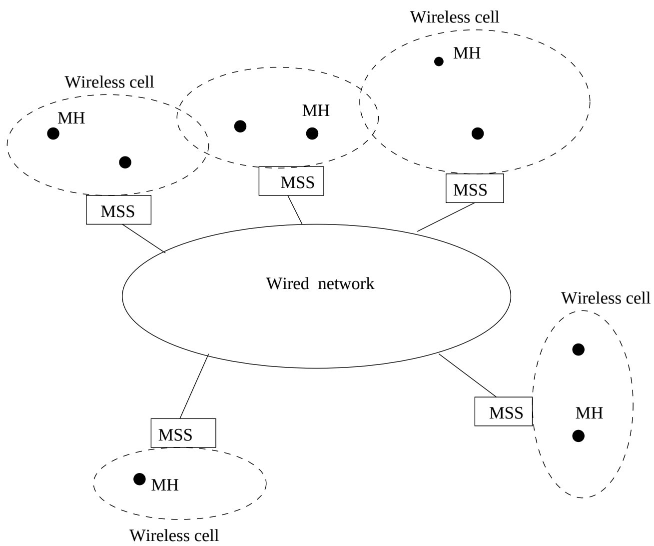

Figure 1: A mobile computing system.

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*. We assume that every message sent in both wired and wireless channels is eventually received, and there are no spurious messages. Figure 1 illustrates a mobile computing system.

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 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 are not suitable for mobile computing systems. We propose an algorithm for causal message ordering in mobile computing systems in which message overhead is small compared to those for static systems and limited resources on mobile hosts are efficiently utilized.

While ordering of messages in distributed systems with static hosts has received wide attention, there has been little work on message ordering in mobile computing systems. Alagar and Venkatesan [6] presented an algorithm that enforces causal ordering among mobile support stations to achieve causal ordering among mobile hosts. Therefore, message overhead of this algorithm is reduced to $O(n^2)$, where n is the number of MSSs. However, some messages are delayed since they violate causal ordering in the MSS's point of view, even though they do not violate causal ordering in the mobile host's point of view. They also proposed a scheme to reduce this extra delay. In this scheme every physical MSS is partitioned into k logical MSSs. Each MH, entering a cell of an (physical) MSS, will be allocated to one of the logical MSSs depending on the load in each logical MSS. Since causal ordering is maintained among logical MSSs, messages to two MHs that belong to different logical MSSs will not have to wait for each other even though both MHs are in the same cell. As a result, message overhead for this scheme is increased to $O(k^2 * n^2)$.

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 still be $O(N^2)$ where N is the number of processes in the system. Furthermore, the size of their message overhead varies when the number of participating processes dynamically changes.

In this paper, we present an algorithm using the concept introduced by Alagar and Venkatesan. However, we can lower the extra delivery delay, and at the same time maintain the size of message overheads at $O(n^2)$. The proposed algorithm is also well-suited for dynamic systems since the size of message overhead is constant, even when the number of participating mobile hosts varies. Our handoff algorithm (the procedure executed when an MH changes cell) allows the MSS to deliver messages that it receives for a new MH sooner than Alagar's handoff algorithm. In their algorithm, the MSS is not allowed to deliver these messages until it receives from the previous MSS a control signal indicating that the handoff algorithm has been complete.

The paper is organized as follows. Section 2 presents the system model and notation used in the paper. Sufficient conditions for causal message ordering in mobile computing systems are presented in Section 3. We present the algorithm in sections 4. Section 5 provides the proof of correctness. We give a comparison between our algorithm and related work in Section 6. Finally, Section 7 concludes the paper.

2 System model

A message passing mobile computation consists of a set of N processes running on mobile hosts, $\{MH_i \mid 0 \leq i < N\}$, and n processes running on mobile support stations, $\{S_j \mid 0 \leq j < n\}$. In general, $N \gg n$. These 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. MH_1 is in the cell of S_1. MH_2 and MH_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*.

We define a relation *immediately precedes* denoted by $\prec_{im}$ between events in a single process as follows: $e \prec_{im} f$ if and only if e immediately precedes f in a process. We use $\prec$ for irreflexive transitive closure[1] of $\prec_{im}$. We say that e locally precedes f if $e \prec f$. Send and delivery events of the same message are related by $\rightsquigarrow$ relation. Since each message is always initiated by an MH and destined to another MH, we can represent a message by a pair (e, f) where e and f are respectively send and delivery events in MH processes of this message. Consider a sample computation shown in Figure 2, $s_1 \rightsquigarrow d_1$, $s_1 \prec_{im} d_2$, and $a \prec e$.

[1] $e \prec f$ iff 1) $e \prec_{im} f$, or 2) $\exists g : (e \prec g \wedge g \prec f)$.

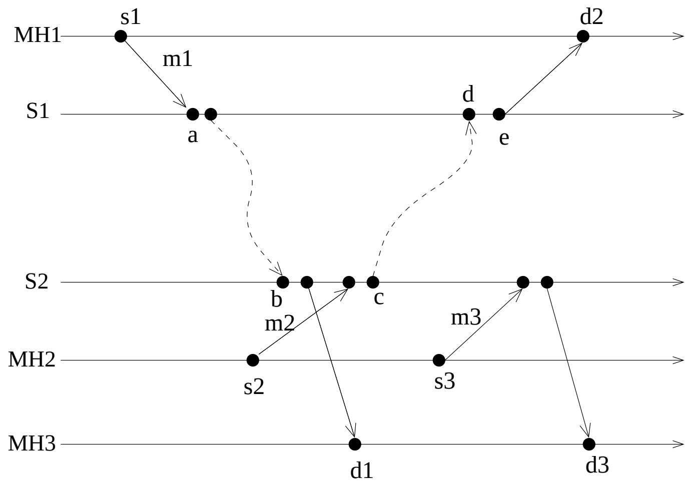

Figure 2: A concrete diagram of a mobile computation

Causal ordering of events in a distributed mobile computation is based on Lamport's "happened before ($\rightarrow$)" relation [16]. The happened before relation is defined as the transitive closure of union of $\prec_{im}$ and $\leadsto$. We also say that a message m_1 causally precedes another message m_2 if the send event of m_1 in MH process causally precedes the send event in MH process of message m_2. Graphically, $m_1 \rightarrow m_2$ if one can follow a path of causality from the send event of m_1 to the send event of m_2 in the abstract diagram. Observe that for any events e, f in distinct processes; if $(e \not\leadsto f) \wedge (e \leadsto f)$, then there is at least one message m sent after e and delivered before f. We say that m is involved in the *causal chain* between e and f.

3 Sufficient Conditions

A mobile computation is causally ordered if the following property holds for any two messages, (s_1, d_1) and (s_2, d_2).

$$s_1 \rightarrow s_2 \Longrightarrow \quad \neg(d_2 \prec d_1) \qquad\qquad (\mathbf{CO})$$

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

Theorem 3.1 *: 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 MH_i to MH_j and message m_2 be sent from MH_k to MH_j. Given $s_1 \rightarrow s_2$, where s_1 and s_2 are respectively the send events of m_1 and m_2 in MH_i and MH_k, we need to show that if C_1, C_2, and C_3 are satisfied, then m_1 and m_2 are delivered to MH_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 $s_1 \rightarrow s_2$, there must be a message path from MH_i

to MH_k via S_i and S_k if S_i is the MSS of MH_i, and S_k is the MSS of MH_k. From C_1 and the fact that $s_1 \rightarrow s_2$, the send event of m_1 in S_i must *causally precede* the send event of m_2 in S_k. Note that this is still true even if MH_i and MH_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, MH_j will deliver m_1 and m_2 in that order. Figure 3 illustrates a causally ordered computation in which all MHs are connected to different MSSs. $\square$

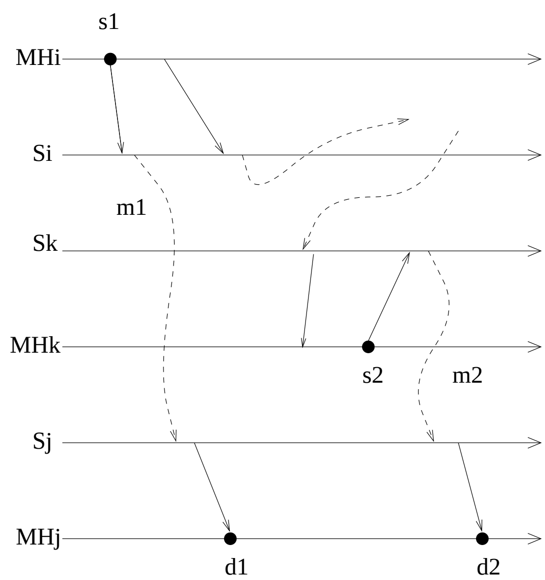

Figure 3: A concrete diagram showing a causally ordered mobile computation.

Sufficient conditions shown in Theorem 3.1 were implicitly used in [6]. For systems with static hosts, Theorem 3.1 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].

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.

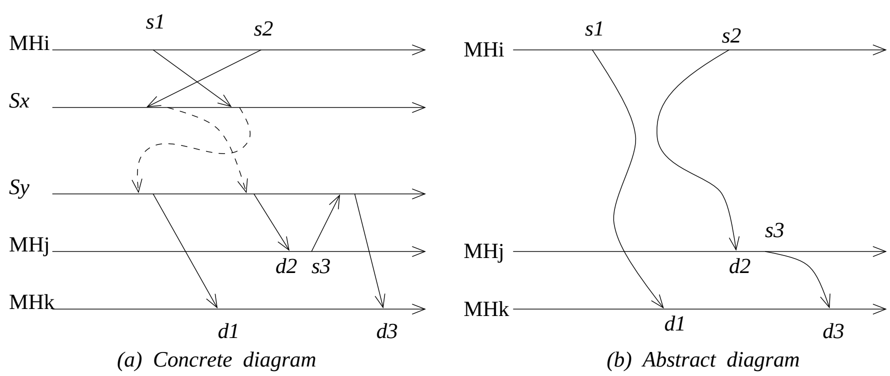

Figure 4: A counter-example to show that C_1, C_2, and C_3 are not necessary.

Since messages must be sent through MSS(s), a message sometimes causally precedes another message in the *MSS's view* but not in the *MH's view*. A message m_1 causally precedes another message m_2 in the MSS's view (denoted by $m_1 \to_s m_2$) if the send event of m_1 in the MSS process causally precedes the send event of m_2 in the MSS process. This definition also holds for the "locally precedes" relation; we use $\prec_s$ to denote the locally precedes relation in the MSS's view.

Let us consider a computation in Figure 5. In this example, MH$_a$ is in the cell of S$_i$, MH$_b$ and

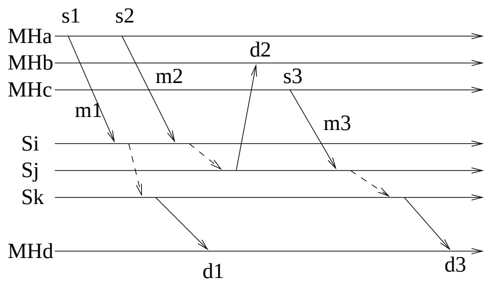

Figure 5: A mobile computation in which $m_1 \to_s m_3$, but $m_1 \not\to m_3$.

MH$_c$ in the cell of S$_j$, and MH$_d$ 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 does not causally precede m_3 in the MH's view. Therefore, m_3 can be delivered to MH$_d$ before m_1 without violating CO. However, m_1 causally precedes m_3 in the MSS's view because the send event of m_1 in S$_i$ causally precedes the send event of m_3 in S$_j$. Hence, $m_1 \to_s m_3$. In this example, $m_1 \prec_s m_2$, and also $m_1 \prec m_2$. Observe that if m causally precedes m' in the MH's view, then m also causally precedes m' in the MSS's view when the channel between each MH and its MSS is FIFO.

We can formally state condition C_2 by using relations $\to_s$ and $\prec_s$ as

$$s_1 \to_s s_2 \Longrightarrow \neg(d_2 \prec_s d_1) \qquad (\mathbf{CO'})$$

The algorithm presented by Alagar and Venkatesan [6] enforces $\mathbf{CO'}$ among MSSs in order to achieve **CO**. This algorithm delays messages that violate $\mathbf{CO'}$ even though they do not violate **CO**. Our goal is to reduce this extra delay, while maintain the size of message overhead in the wired network to $O(n^2)$. To reduce the extra delay, we propose another property denoted by $\mathbf{CO''}$ to be implemented among MSSs.

$$((s_1 \to s_2) \vee (s_1 \prec_s s_2)) \Longrightarrow \neg(d_2 \prec_s d_1) \qquad (\mathbf{CO''})$$

Consider an example given in Figure 6, message m_1 does not causally precede m_3 in the MH's view, but it does in the MSS's view. Our algorithm allows m_3 to be delivered by S$_k$ before m_1. If we implement $\mathbf{CO'}$, then m_3 will be delayed until m_1 has been delivered.

From the fact that $s_1 \to s_2 \Longrightarrow ((s_1 \to s_2) \vee (s_1 \prec_s s_2))$ and wireless channels are FIFO, causal message ordering in mobile computations is also achived by implementing $\mathbf{CO''}$.

4 Algorithm

Since the FIFO channel between each MH and its MSS can be implemented simply by using sequence numbers, we will focus only on the algorithm to implement $\mathbf{CO''}$ among MSSs. This algorithm

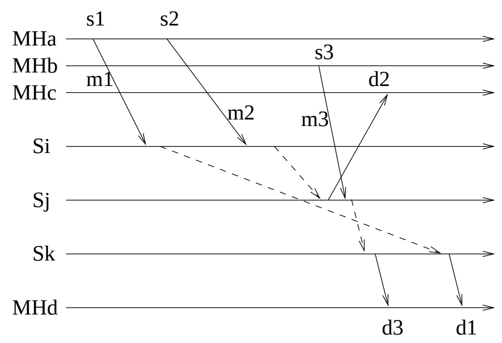

Figure 6: An example of our implementation

will be called *static algorithm*. However, static algorithm alone is not sufficient to maintain causal message ordering when MHs change cells. The procedure executed when an MH changes cell is referred to as the *handoff algorithm*.

Static algorithm

Our static algorithm is based on the algorithm proposed by Raynal et al. [19]. We assign an integer matrix, M_i, of size $n \times n$ to each mobile host MH$_i$ in the system. All entries in M_i are initially zero. In our algorithm, M_i is stored at the MSS of MH$_i$. Therefore, each MSS maintains a set of matrices belonging to MHs in its cell. Let $\mathcal{M}_i$ denote the set of matrices maintained by S$_i$. The algorithm is given in Figure 7.

Whenever an MH$_a$ wants to send a message to MH$_b$, if MH$_a$ and MH$_b$ are in different cells, then MH$_a$ must send this message to its MSS (S$_i$). Once S$_i$ receives the message, it appends the matrix M_a to the message and sends it to S$_j$ the MSS of MH$_b$. Then, S$_i$ increments the entry $M[i,j]$ of each matrix in $\mathcal{M}_i$ by 1. Once S$_j$ receives the message, it checks if the message is *deliverable* by determining the following constraint:

$$\forall x : M_b[x,j] \geq M_a[x,j]$$

If the message is deliverable, S$_j$ sends the message to MH$_b$ and increments the entry $M[i,j]$ of each matrix in $\mathcal{M}_j$ by 1. Then, S$_j$ takes a component-wise maximum between M_b and M_a and assigns it to M_b. Note that only the matrix corresponding to the receiving MH is updated by the result of component-wise maximum operation. If the message is not deliverable, S$_j$ keeps it in the buffer until it is deliverable.

In the case that MH$_a$ and MH$_b$ are in the same cell (S$_i$), when the message arrives at S$_i$, S$_i$ increments the entry $M[i,i]$ of each matrix in $\mathcal{M}_i$ by 1. Then, the component-wise maximum operation between M_a and M_b is performed, and the result is assigned to only M_b. Observe that each message sent/received within a cell will be delivered to the destination as soon as it arrives at the MSS. We will prove this does not violate $\mathbf{CO}''$ in the next section.

From the sending routine in program text, every time S$_i$ sends a message to any support station S$_x$ on behalf of any of the MHs in its cell, the entry $M[i,x]$ of any matrices in $\mathcal{M}_i$ is incremented by one. Accordingly, for any MH$_u$ served by S$_i$, $M_u[i,x]$ records the number of messages sent from S$_i$ to S$_x$ by any MHs in S$_i$ delivered to any MHs in S$_x$, $\forall x \in \{1,\ldots,n\}$. Similarly, from the receiving routine, we know that $M_u[x,i]$ records the number of messages sent from S$_x$ to S$_i$ by any MHs in S$_x$ delivered to any MHs in S$_i$, $\forall x \in \{1,\ldots,n\}$. From above, we can also conclude that for all i,

S_i::

var

$\mathcal{M}_i$: a set of matrices maintained by S_i.

/* *sending routine* */

(S) Upon sending a message m on behalf of MH_s to MH_r

(S.0) If (S_i is also the MSS of MH_r)

(S.1) $\forall x \in \mathcal{M}_i$: M_x[i,i] ++

(S.2) M_r := max(M_s,M_r)

(S.3) send m to MH_r

(S.4) else

(S.5) append matrix M_s to m

(S.6) send m to S_j /* *the MSS of MH_r* */

(S.7) $\forall x \in \mathcal{M}_i$:M_x[i,j] ++

/* *receiving routine* */

(R) Upon receipt of a message m on behalf of MH_r from S_j

/* *Let M_u be the matrix attached with message m* */

(R.0) if ($\forall$x : M_r[x,i] $\geq$ M_u[x,i])

(R.1) send m to MH_r

(R.2) $\forall x \in \mathcal{M}$: M_x[j,i] ++

(R.3) M_r := max(M_r,M_u)

(R.4) else

(R.5) keep m in the buffer until it is *deliverable*.

Figure 7: The static algorithm for an MSS S_i

the i^{th} row of all the matrices maintained by S_i are identical. Likewise, the i^{th} column of all the matrices maintained by S_i are also identical. Since the result of max operation (R.3) is assigned only to the matrix of the destination of the message, information represented by other entries in M_u is different from above. The entry $M_u[x, y]$ represents the number of messages sent from S_x to S_y as known by MH_u. It is noted that if there is exactly one mobile host per support station, the algorithm reduces to that of [19].

Handoff algorithm

To preserve causal message ordering, handoff algorithm is executed every time an MH changes cell. Each MSS is required to update its local knowledge about the current location (the address of the new MSS) of any moving MHs. We assume that initially each MSS knows the exact location of each MH. Many strategies for this update have been proposed (see for example [7, 9, 15]). Our handoff algorithm can be used with any existing update strategies.

Because of asynchronous nature of communication channels in the wired network, it is possible that a message is sent to a MSS (according to the local knowledge of the sender) in which the MH receiver of this message has already left the cell. Therefore, handoff algorithm is required to forward this message to the new MSS and deliver it to the MH with respect to causal ordering. In particular, a message sent to an MH that just enters a cell needs a special care from the MSS of this cell.

```
S_i::
(I.1)    Upon receipt of handoff_start(k) from S_j.
             record the new location of MH_k.
             send enable(M_k^i) to S_j.
(I.2)    Upon receipt of a message m for MH_k after handoff_start(k) is received.
             if (m is deliverable)
                 send forward(m,M_k^i) to S_j.
(I.3)    Upon receipt of all flush(k) signals.
             send handoff_done(M_k^i) to S_j.
             terminate the handoff algorithm for MH_k.
```

Figure 8: The handoff algorithm for S_i

Moreover, checking constraint (R.0) in order to deliver a message m to an MH_k in the new cell is not sufficient to maintain causal ordering since there could be messages that causally precede m sent to the previous MSS, and not yet delivered to MH_k. In addition to the requirements given above, handoff algorithm should also enable any MH entering a new cell to send and receive messages through the new MSS as early as possible. Our handoff algorithm satisfies these requirements by using control messages referred to as *signals*.

We now briefly discuss the executions of MSS S_i and S_j when an MH_k moves from S_i to S_j. The algorithm for S_i and S_j are given in Figure 8 and Figure 9, respectively. Once MH_k enters the cell of S_j, it sends a *greeting* signal to S_j to inform S_j of its presence. On receiving *greeting* signal, S_j sends a *handoff_start(k)* signal to S_i to inform S_i about the new location of MH_k. Once S_i receives *handoff_start(k)*, it makes a copy of matrix M_k, and sends the copy along with the *enable* signal to S_j. We denote matrix M_k maintained by S_i during the execution of the handoff algorithm by M_k^i. When S_j delivers *enable* and properly updates matrix M_k, S_j can start sending messages on behalf of MH_k. We denote matrix M_k at S_j during the execution of the handoff algorithm by M_k^j. We will later explain how S_j updates this matrix.

If S_i receives a message m for MH_k after *handoff_start(k)* signal, S_i checks whether or not m is deliverable by using static algorithm and M_k^i. When m is deliverable, S_i attaches m and matrix M_k^i to a *forward* signal, and sends it to S_j. To terminate handoff algorithm, we must ensure that all messages for MH_k sent from other MSSs (except S_j) to S_i have been received. Therefore, we require that a *flush*$_k$ signal is sent from each MSS when it updates the local knowledge of the current location of MH_k. On receiving a *flush*$_k$ signal from any MSS S_q, S_i knows that all the messages for MH_k sent from S_q has been arrived at S_i, and S_q will never send messages for MH_k to S_i again. We can implement *flush* signals by using *forward-flush* primitive introduced by Ahuja [2]. Once all *flush(k)* signals are received, and there are no messages for MH_k in the buffer of S_i, S_i sends a *handoff_done(k)* signal to S_j, discards matrix M_k^i, and then terminates handoff algorithm for MH_k. S_j also terminates its handoff algorithm when it receives *handoff_done(k)*.

If S_j receives a message m before *enable* signal is delivered, m will be kept in the buffer. As we mentioned earlier, the j^{th} row of any matrices maintained by S_j are all identical. The same also holds for all the j^{th} columns. For convenience, we respectively refer to these values by row_j and col_j. To preserve causal ordering, we require that both *forward* and *enable* signals are checked

```
S_j::
(J.1)     Upon receipt of greeting from MH_k.
               send handoff_start(k) to S_i.
(J.2)     Upon receipt of a message m for MH_k before receiving handoff_done.
               if ((∀x : M_k^j[x,j] ≥ M_u[x,j]) and (∀y : M_k^j[y,i] ≥ M_u[y,i]))
                    send m to MH_k.
               else
                    keep m in the buffer.
(J.3)     Upon receipt of enable( M_k^i ) or forward ( m, M_k^i ) from S_i.
               if (∀x : col_j[x] ≥ M_k^i[x,j])
(J.3.1)             if (enable)
                       replace column j^th of M_k^i by col_j
                       replace row j^th of M_k^i by row_j
                       M_k^j := M_k^i
                       add M_k^j into 𝓜_j
                       ∀z ∈ 𝓜_j :   M_z[i,j] ++
                       S_j can start sending messages on behalf of MH_k.
(J.3.2)             if (forward)
                       ∀z ∈ 𝓜_j :   M_z[i,j] ++
                       M_k^j := max (M_k^i, M_k^j)
                       send m to MH_k.
               else
                    keep signal in the buffer.
(J.4)     Upon receipt of (handoff_done(k)).
               terminate the handoff of MH_k.
```

Figure 9: The handoff algorithm for S_j

whether or not they are deliverable (step (J.3)).

When *enable* is deliverable, the j^{th} column and j^{th} row of M_k^i are replaced by col_j and row_j, respectively (step (J.3.1)). At this point, S_j can begin sending messages on behalf of MH_k. When a *forward* signal is deliverable, matrix M_k^j is updated by the same operations used in the static algorithm, and S_j then sends message m to MH_k (step (J.3.2)).

5 Proof of Correctness

In this section, we prove that static and handoff algorithms together ensure causal message ordering in mobile computations (*safety*). Furthermore, we must show that every message sent is eventually delivered to its destination (*liveness*).

5.1 Proof for Static Algorithm

Before proceeding with the proof, we need to show some properties regarding the information carried in each matrix. Notation used in the proof will be defined as follows: for any send event e in process MH_q, $e.M_q$ denotes the matrix attached to the message sent from the MSS to which MH_q

is connected. If the destination MH process is also in the same cell as MH_q, then $e.M_q$ denotes the matrix of MH_q at the state in the MSS process that message e arrives.

Lemma 5.1 *Let s_1 and s_2 be the send events in MH_1 and MH_2, respectively. If S_i is the MSS of MH_1 and S_j is the MSS of the destination MH of message s_1, then (including $i = j$)*

$$s_1 \to s_2 \Longrightarrow s_1.M_1[i,j] < s_2.M_2[i,j]$$

Proof: We prove the lemma by induction on k, the number of messages in the causal chain between s_1 and s_2.
Base($k = 0$): Since there is no message in the causal chain, $s_1 \prec s_2$. By the FIFO property, we know that message s_1 must arrive at S_i before message s_2. From step (S.1) in the program text, we know that $s_1.M_1[i,j] < s_2.M_2[i,j]$.
Induction($k > 0$): Let there be k messages in the causal chain, and $e_1, \ldots, e_k$ denote the send events of these messages. Assume that message e_k is sent by MH_k. By induction hypothesis,

$$s_1.M_1[i,j] < e_k.M_k[i,j] \tag{1}$$

Since the k^{th} message must be delivered by MH_2 before s_2, from step (R.3), we know that

$$e_k.M_k[i,j] \leq s_2.M_2[i,j] \tag{2}$$

The lemma follows from (1) and (2). □

Theorem 5.2 *(Safety): The static algorithm ensures causal message ordering in the absence of mobility.*

Proof: It is sufficient to show that the algorithm ensures **CO**$''$, since **CO** can be achieved by enforcing **CO**$''$.

- (A) : $s_1 \to s_2 \Longrightarrow \neg(d_2 \prec_s d_1)$
  Let (s_1, d_1) be a message sent from MH_1, and (s_2, d_2) sent from MH_2. If these two messages are not sent to the same MSS process, then (A) holds trivially. Otherwise, assume that both messages are sent to MSS S_j. We prove the theorem by induction on k, the number of messages delivered by S_j. (A) clearly holds in the *base case ($k = 0$)* . By induction hypothesis, we assume that (A) holds for the first $k-1$ messages, and show that the delivery of k^{th} message maintains (A). We will show that $(s_1 \to s_2) \wedge (d_2 \prec_s d_1)$ leads to a contradiction. Assume that d_2 corresponds to the k^{th} message, and message d_1 is not yet delivered.

  Assume that S_i is the MSS of MH_1, and message s_2 is intended for MH_d which is in the cell of S_j. Let z be the state in S_j in which k^{th} message is delivered. Since $s_1 \to s_2$, it follows from Lemma 5.1 that

$$s_1.M_1[i,j] < s_2.M_2[i,j] \tag{3}$$

  Note that Equation 3 still holds even if $j = i$. At the state z, process S_j has delivered at most $M_d[i,j]$ messages sent via S_i and S_j to MH_d (some messages may belong to other MHs in the cell of S_j). We have assumed that message s_1 has not been delivered. Further, no message sent after s_1 has been delivered so far since that would violate the induction hypothesis. As mentioned, $s_1.M_1[i,j]$ denotes the number of messages sent from S_i to S_j by any MHs in the cell of S_i. Therefore, $M_d[i,j] = s_1.M_1[i,j]$. From (3), it implies that $M_d[i,j] < s_2.M_2[i,j]$. If MH_2 and MH_d are not in the same cell, from condition (R.0), we know that the delivery of message s_2 at S_j is not enabled at this point. But earlier, we had assumed that message s_2 is delivered at z, a contradiction.

Now we show that even if MH_2 and MH_d are in the same cell, $(s_1 \rightarrow s_2) \wedge (d_2 \prec_s d_1)$ leads to a contradiction. By FIFO property, $d_2 \prec_s d_1$ implies that message m_2 must be delivered by S_j before m_1. Since $s_1 \rightarrow s_2$, the last message in the causal chain between s_1 and s_2 must be delivered at S_j before m_2, otherwise there exists a cycle. This implies that there exists a message in the causal chain delivered by S_j before m_1. It leads to a contradiction since by induction hypothesis, we know that (A) holds for the first $k-1$ messages delivered by S_j.

- (B) : $s_1 \prec_s s_2 \Longrightarrow \neg(d_2 \prec_s d_1)$.
  To prove (B), we have to show that the static algorithm implements FIFO communication between MSSs. This is obvious since each MSS S_i increments the entry (i,j) of all the matrices in $\mathcal{M}_i$ every time it sends a message to S_j. S_j delivers the message only when (R.0) holds, and S_j then increments its entry (i,j) of all the matrices in $\mathcal{M}_j$.

□

Theorem 5.3 *(Liveness): Every message sent is eventually delivered to its destination.*

Proof: Let $C_j(G,z)$ be the set of messages sent to S_j, in the channel in the consistent cut G (which includes the state z in S_j). Let $M = min(C_j(G,z))$ represent the set of messages minimal in $C_j(G,z)$ with respect to $\rightarrow_s$. Let $\mu \in M$ and μ be sent from MH_u to MH_k. To prove the theorem, it is sufficient to show that μ is deliverable. We show the contrapositive, that is, if μ is not deliverable at z, then $\mu \notin M$.

The fact that μ is not deliverable at z, implies that $\exists y : M_k[y,j] < M_u[y,j]$ (step (R.0)). Let $M_u[y,j] = n$. This means that there are n messages sent from S_y to S_j as known by MH_u. Note that these messages can be sent to any MHs in the cell of S_j. If all these messages have been delivered to S_j, then M_k[y,j] = n (step (R.2)), a contradiction. Otherwise, there exists a message which causally precedes μ as known by MH_u, and is not delivered before state z. Thus, $\mu \notin M$. □

5.2 Proof for Handoff Algorithm

Next, we show that our handoff algorithm preserves safety and liveness properties when MHs change cells.

Theorem 5.4 *(Safety): The handoff algorithm preserves causal message ordering when MHs change cell.*

Proof: It is sufficient to prove that the handoff algorithm preserves **CO**$''$ when MHs changes cells. We follow the same approach used in Theorem 5.2. Assume that MH_k moves from S_i to S_j, and message s_1 is sent by MH_1 in the cell of S_p. There are two cases to be considered: (1) s_1 is received by S_i after MH_k has left, and (2) message s_1 is received by S_j before S_j receives *handoff_done* signal. Again, we show that $(s_1 \rightarrow s_2) \wedge (d_2 \prec_s d_1)$ leads to a contradiction.

For (1), we know that $M_k^j[p,i] \leq s_1.M_1[p,i]$ since only messages intended for MH_k are forwarded from S_i to S_j, and matrix M_k^j is updated (step (J.3.2)), whenever S_j delivers each of these forwarded messages. From Lemma 5.1, $s_1.M_1[p,i] < s_2.M_2[p,i]$. Therefore, it implies that $M_k^j[p,i] < s_2.M_2[p,i]$. By step (J.2), message s_2 is not deliverable, a contradiction.

For (2), we know that $M_k^j[p,j] = s_1.M_1[p,j]$ since j^{th} row and j^{th} column of M_k^i are initially replaced by row_j and col_j, and M_k^i is assigned to M_k^j (step (J.3.1)). Then, M_k^j is updated in step (J.3.2) whenever a forwarded message is delivered. Again, from Lemma 5.1, we know that $M_k[p,j] < s_2.M_2[p,j]$. From step (J.2), message s_2 is not deliverable, a contradiction. □

Theorem 5.5 *(Liveness): The handoff algorithm preserves the liveness property when MHs change cells.*

Proof: We have to show that any message μ for MH_k delayed by step (J.2) in the handoff algorithm is eventually delivered to MH_k. If μ is not deliverable, it implies that (1) $\exists y : M_k[y,j] < M_u[y,j]$ or, (2) $\exists x : M_k[x,i] < M_u[x,i]$. If μ is not deliverable due to (1), it means that there exists a message sent to S_j which causally precedes μ as known by MH_u (the sender of μ). In this case, we know that μ is eventually delivered to MH_k because of the liveness property of the static algorithm, and the fact that matrix M_k^j is updated by steps (J.3.1) and (J.3.2).

If μ is not deliverable due to (2), it means that there exists a message m' sent to S_i which causally precedes μ as known by MH_u. Note that m' may or may not be intended for MH_k. If m' is intended for MH_k, then m' will eventually be forwarded to S_j, and delivered to MH_k. Otherwise, *handoff_done* signal will terminate the handoff algorithm on S_j and μ will eventually be delivered to MH_k due to the liveness property of the static algorithm. □

6 Comparison to Previous Work

The proposed algorithm has less delivery delay than that of Alagar and Venkatesan, while its message overhead is still $O(n^2)$. By enforcing $\mathbf{CO'}$, a message m_2 is always delayed for another message m_1 if and only if $m_1 \rightarrow_s m_2$. On the other hand, our algorithm delays message m_2 for message m_1 when the matrix carried by m_2 contains information of the send of m_1. The computation given in Figure 6 is an example when a message has to be delayed by Alagar's algorithm, but does not by our algorithm.

Our memory overhead in each MSS is $O(k * n^2)$, where k is the number of MHs currently in the cell of the MSS. Even though this overhead is higher than that of Alagar's algorithm, it can be easily accommodated by MSSs due to their rich memory resources.

Our handoff algorithm allows the new MSS to deliver messages that it receives on behalf of the MH that just enters the cell whenever the delivery constraint (J.2) is satisfied. In Alagar's handoff algorithm, the new MSS has to wait until it receives *handoff_done* signal from the previous MSS. Our handoff algorithm enables the new MSS to send messages on behalf of the newly arrived MH as soon as Alagar's algorithm.

Prakash's algorithm [18] is not suitable for dynamic systems because the structure of information carried by each message in their algorithm depends on the number of participating processes. Each message among MSSs in our algorithm carries an integer matrix of size $n \times n$, which is constant, even when the number of MHs dynamically changes. Prakash's algorithm, however, incurs no extra delay in message delivery.

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

Algorithm	Message overhead	Extra delay in message delivery	Well-suited for dynamic systems
Alagar Venkatesan	$O(n^2)$	High	Yes
Prakash et al	$O(N^2)$	No	No
Skawratananond Garg	$O(n^2)$	Low	Yes

N: the number of MHs.
n: the number of MSSs.

7 Conclusions

We have presented a lightweight algorithm for causal message ordering in mobile computing systems. Our algorithm requires $O(n^2)$ of memory and message overheads, where n is the number of MSSs. The low message overhead is obtained in exchange of the extra delay in message delivery. However, we reduce this delay by assigning an $n \times n$ matrix to each MH instead of having all MHs in a cell shared the same matrix. Furthermore, information carried by each message in our algorithm is constant, whereas the number of MHs varies. Therefore, our algorithm scales well and is also suitable for systems in which the number of MHs dynamically changes.

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] Sridhar Alagar and S. Venkatesan, Causal Ordering in Distributed Mobile Systems, In *Proceedings of Workshop on Mobile Computing Systems and Applications*, December 1994, pp. 169-174.

[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.