A protocol for causally ordered message delivery in mobile computing systems *

Li-Hsing Yen, Ting-Lu Huang and Shu-Yuen Hwang

Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan, R.O.C.

There is a growing trend in developing applications for mobile computing systems in which mobile host computers retain their network connections while in transit. This paper proposes an algorithm that enforces a useful property, namely, causal ordering, that delivers messages among mobile hosts. This property ensures that causally related messages directed to the same destination will be delivered in an order consistent with their causality, which is important in applications that involve human interaction such as mobile e-mail and mobile teleconferencing. Such applications are envisioned by the proponents of Personal Communications Services (PCS). Without this property, users may receive and read original messages and the corresponding replies out of order. Our algorithm, when compared with previous proposals, provides an alternative with a low handoff cost, medium message overhead, and low probability of unnecessary inhibition in delivering messages.

1. Introduction

There has been considerable interest recently in adapting algorithms designed for conventional distributed systems to mobile computing environments. This work is not trivial because the designs of conventional distributed systems do not take into account many characteristics of mobile systems such as changeable physical network connections, resource constraints on mobile hosts, and the limited bandwidth of wireless links [6]. To cope with changeable physical network connections, there should be some appropriate way of handling host migrations and disconnections; to overcome resource constraints and limited bandwidth, the computation load and communication overhead on mobile hosts should be kept low [6,10]. Reported contributions toward this research direction include distributed mutual exclusion algorithms [7], checkpointing algorithms [2,19], atomic multicast protocol [1,5], and *causally ordered message-delivery* (COMD) algorithms [3]. In this paper, we consider the problem of preserving the COMD property among mobile hosts.

In systems in which the COMD property is preserved, messages directed to the same destination are delivered in an order consistent with their causality. The causality under consideration is determined by the *happens-before* relationship [12] but restricted to message sending and receiving events. In other words, if the sending event of a message happens before that of another message, the former message should be received before the latter message is. Causally ordered communication is considered an important facility for constructing reliable distributed systems [4,8,9,15]. It can also be used to maintain consistency among replicated data located at different sites [8]. For a clear understanding of the importance of COMD in mobile environments, consider a teleconferencing application in which two or more persons can hold a conversation via their mobile computers. We assume that each participant in the conversation can send messages and respond to others at any time, and also that an original message and its responses may be sent to a number of participants. Suppose now that user A sends out a message and that, after receiving this message, user B issues a response. The COMD property ensures that no participant can receive the response from B before receiving the original message from A. Without COMD, a third party may receive a comment on some idea before receiving the message expressing the original idea. In that case, the idea behind the conversation can be misinterpreted.[1]

Personal Communications Services (PCS) has the ultimate goal to provide truly personal, timely communication services to users through portable handsets.[2] Information being exchanged in a PCS network includes voice, data, image, and even video [13]. COMD property is not only important in teleconferencing applications, but also a communication primitive for computation tasks that involve more than two mobile hosts. These applications, which are envisioned by the proponents of PCS, should be attractive to users of cellular phone systems.

Many COMD algorithms have been designed for stationary distributed systems (e.g., [8,16,20–22]). Alagar and Venkatesan [3] proposed three extensions of the COMD algorithm in [20] for mobile computing systems. In their first algorithm, the COMD property is explicitly maintained among all mobile hosts (MHs). To reduce computation and communication loads on mobile hosts, this algorithm stores data structures relevant to causal ordering in mobile sup-

* This research has been supported by the National Science Council, ROC, under grant NSC 85-2213-E-009-062.

[1] Most existing teleconferencing programs are structured on the client-server model, where one server serializes all conversations among clients. Here we assume a peer-to-peer model, where one mobile host can communicate directly with all others.

[2] Portable handsets in this paper are referred to as mobile hosts to emphasize their ability of computing.

port stations (MSSs), and the algorithm is executed by the MSSs on behalf of the MHs. The message overhead is proportional to the square of the number of mobile hosts, and the algorithm is not graceful in handling host disconnections/connections. The advantage of this algorithm is that the procedure for handling host migrations is simple.

In their second algorithm, the COMD property is explicitly maintained among MSSs, which is sufficient to enforce the COMD property among MHs if each communication channel between an MSS and an MH in the MSS's cell provides FIFO-order delivery and no MH ever moves. To deal with problems associated with mobility, this algorithm is equipped with a complicated module for handling host migrations. Message overhead in this algorithm is proportional to the square of the number of MSSs, and host disconnections/connections do not pose any problem. However, this approach enforces a stronger ordering of messages so that occasional unnecessary delays in delivering messages may occur. To reduce this probability, Alagar and Venkatesan proposed a third algorithm, which is a variant of the second one. Each MSS is partitioned into k logical MSSs. When an MH enters the cell of a physical MSS, it is allocated to one of the k MSSs, and the COMD property is maintained among all logical MSSs. As k increases, the possibility of unnecessary delays decreases, but the message overhead and the cost in handling host migrations increases.

The computation load and the communication overhead on mobile hosts are both low in all these algorithms, though the message overhead varies. However, none of these algorithms handles both mobility and disconnection in an efficient way. The first algorithm is not scalable for frequent disconnections, while the second and the third algorithms require a time-consuming procedure to handle host migrations.

In this paper, we propose another COMD algorithm for mobile computing systems. This algorithm is basically a compromise between the first and the second algorithms mentioned above. Message overhead in our algorithm is proportional to the product of the number of MHs times the number of MSSs. Generally, this overhead is lower than that of the first algorithm and higher than that of the second algorithm. It is easier to handle host migrations using our algorithm than when using either of the others. Our algorithm is more scalable than the first when considering host disconnections, and it can also reduce the probability of unnecessary inhibition inherent in the second algorithm.

2. Problem definition

2.1. Model of a mobile computing system

We adopt the model given in [6] as the extraction of the underlying execution environment for our algorithm. This model consists of three major components: a wired network, mobile hosts (MHs), and mobile support stations (MSSs). They are discussed in detail as follows (see also figure 1).

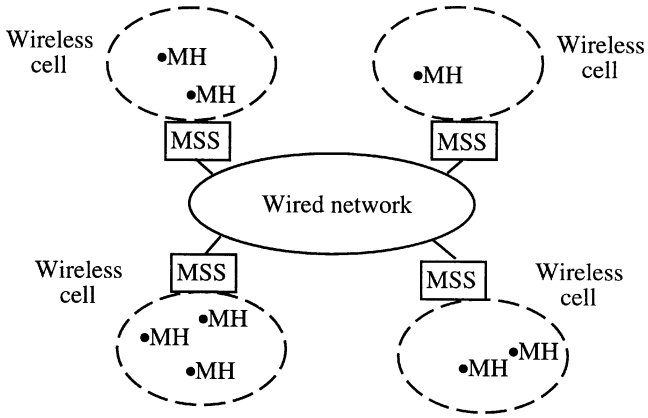


Figure 1. The model of a mobile computing system.

- *Wired network*: A data network that connects all MSSs. Every node in this network can communicate with any other through wired channels.
- *MHs*: An MH is a computer that can move while retaining communication with other computers by sending and receiving messages through wireless channels. An MH can disconnect itself from the network at any time and reconnect at a later time.
- *MSSs*: MSSs form the infrastructure of this system model and can communicate directly with MHs over wireless channels. Due to physical constraints, the geographical area an MSS's wireless signal can cover is limited. We call such a geographical area the *cell* of the MSS. Only MHs resident in an MSS's cell can directly communicate with that MSS. At any given time, an MH is assumed to be within the cell of at most one MSS, which is called its local MSS while it remains there. All messages directed to an MH h are first routed through nodes in the wired network to h's local MSS, then sent to h via a wireless channel. Similarly, all wireless messages transmitted by an MH are first received by the MH's local MSS, and then routed to their destinations. Thus an MSS serves as an access point to the wired network for all MHs currently in its cell.

Every MSS periodically broadcasts a beacon signal [11], which carries the MSS's identification, to all the MHs in its cell. Upon receiving the beacon, an MH determines which MSS it should communicate with at that time. However, an MH can receive more than one beacon signal at the same time when it is crossing the boundary between two or more cells. Nevertheless, we assume that an MH can always choose one (and only one) MSS as its local MSS. The MH can accomplish this by collecting and measuring the quality (such as signal strength) of received beacons. Thus, when an MH switches from one MSS to another, a *hand-off* procedure is performed. The MH first sends a *register* message, which carries the identification of the

previous local MSS, to the new local MSS. On receiving this message, the new local MSS informs the previous local one of the migration of this MH. In response to this message the previous MSS will transmit relevant information associated with the MH to the current MSS.

2.2. Causal ordering of message delivery

The atomic operation that changes the state of a process is defined as an *event*. Three types of events are considered to occur in traditional distributed systems: the *sending* of messages, the *receipt* of messages, and internal events [14]. What constitutes an internal event depends on the context of systems and is irrelevant to the definition of the COMD property. We assume that there is no perfectly accurate clock available in any MH. With such a premise, the temporal ordering of events occurring in MHs can only be a partial relation. One of such is the "happens-before" relation defined by Lamport [12]. The "happens-before" relation (denoted by $\rightarrow$) on the set of events is the smallest transitive relation satisfying the following conditions:

(1) if a and b occur in the same process and if a comes before b then $a \rightarrow b$;

(2) if a is the sending of message m and b is the receipt of m, then $a \rightarrow b$.

Let *sent*(m) be the event that corresponds to the sending of message m and *recv*(m) be the event that corresponds to the receipt of m. The causal ordering of message delivery is obeyed if, for any two messages m and m' that have the same destination, *sent*(m) $\rightarrow$*sent*(m') implies *recv*(m) $\rightarrow$*recv*(m').

3. The algorithm

3.1. Preliminaries

Our COMD algorithm for mobile computing systems is based on the contribution of Raynal et al. [20]. Their algorithm, hereafter referred to as the RST algorithm, is based on message counting and relies on a deferred receiving technique that defers the receiving of a message from the delivering of the same message. A message is said to be *received* by a site when it arrives at that site. A message is said to be *delivered* by a site when it is formally accepted and disposed of by the associated application running at that site. All received but undelivered messages are buffered in a message queue. A distributed computing system, in which the COMD property does not hold with respect to sending and receiving events, can employ a COMD algorithm to enforce the COMD property with respect to sending and delivering events. That is, let *deliv*(m) represent the event of delivering message m, a COMD algorithm can ensure that *sent*(m) $\rightarrow$ *sent*(m') always implies that *deliv*(m) $\rightarrow$*deliv*(m').

The operation of the RST algorithm is as follows. Suppose that there are n sites in a distributed computing system. Each site S_i maintains an $n \times n$ matrix SENT$_i$. The jth row of SENT$_i$, SENT$_i[j,k]$, in which $1 \leqslant k \leqslant n$, represents the number of messages site S_j has sent to site S_k, and is known by S_i. Every message transmitted by S_i is tagged with the content of SENT$_i$ when it is issued. SENT$_i[i,j]$ is also incremented by one after a message directed to S_j has been sent by S_i.

Each site S_i in the system also maintains an n-entry vector DELIV$_i$. The jth entry of DELIV$_i$ represents the number of messages sent by S_j that have been delivered by S_i. When a message m is received at S_i, S_i compares the content of DELIV$_i$ with the content of the matrix tagged m. Suppose that, without loss of generality, m was sent by S_j and received at S_i. Let $m.S$ represent the matrix content tagging message m, m will then be delivered if DELIV$_i[k] \geqslant m.S[k,i]$ for all k. A message m satisfying this condition means that all causally preceding messages, mentioned in $m.S$, have been delivered to S_i, so m can be delivered without violating the COMD property. After m is delivered, DELIV$_i[j]$ as well as SENT$_i[j,i]$ is incremented by one, and then each element in SENT$_i$ is set to the maximum of its original value and the value of the corresponding element in $m.S$. So the delivery of a message effectively informs the receiving site about the content of the sending site's SENT matrix.

3.2. Restructuring the RST algorithm used in mobile systems

We now review the RST algorithm for mobile computing systems. A straightforward approach is to have each MH involved in the execution of the RST algorithm (actually, the first algorithm proposed by Alagar and Venkatesan takes this approach). Although this approach is feasible, it requires that each message carries a header whose size is O(n_h^2), where n_h is the number of MHs in the system. This overhead is costly since the number of MHs in a mobile environment is typically enormous. Furthermore, due to the disconnections and reconnections of MHs, either the associated arrays and vectors should all be of maximal size, or there should be a way to dynamically change the structure of the associated data. Neither approach is scalable when the number of MHs is enormous and the frequency of disconnections and reconnections is high. For these reasons, we take a variant approach that only records the number of messages sent from each MSS to each MH. The message overhead in our approach is thus only O($n_s \times n_h$), where n_s is the number of MSSs.

Our approach involves using three modules. The main module is presented in this subsection. The handoff module, which handles handoff procedures, is discussed in the next subsection. Finally, in section 3.4, we introduce our method for handling MH's disconnections. We assume that there is already a reliable routing protocol that eventually routes each message m destined for MH h_i to his cur-

rently local MSS. Such a protocol can be found in [11,17]. We do not assume that this protocol provides FIFO-order message delivery between pairs of MSSs. However, the wireless communication channel between each MH and its local MSS is assumed FIFO-ordered.

Let n_s be the number of MSSs and n_h be the number of MHs in a mobile computing system. We assume that each MH as well as each MSS has a unique identifier and, without loss of generality, that $\{h_1, h_2, \ldots, h_{n_h}\}$ and $\{s_1, s_2, \ldots, s_{n_s}\}$ represent the set of all MHs and the set of all MSSs, respectively. Each MSS s_i maintains an $n_s \times n_h$ matrix called MSS_SENT$_i$. The jth row of MSS_SENT$_i$, MSS_SENT$_i[j,k]$, in which $1 \leqslant k \leqslant n_h$, denotes the number of messages s_i knows to have been sent by MSS s_j to MH h_k. Initially, all entries in the MSS_SENT$_i$ arrays are set to zeros for all i. Each message m transmitted by MH h_i is first received by h_i's local MSS, say, s_l. s_l then tags m with s_l's sending matrix, MSS_SENT$_l$, and directs m to its destination, say, MH h_j. Each MSS does such a forwarding service in a first-come-first-serve manner. After doing that, MSS_SENT$_l[l,j]$ is incremented by one.

Each MH h_i in the system is associated with an n_s-entry vector, MH_DELIV$_i$. The jth entry in MH_DELIV$_i$, MH_DELIV$_i[j]$, denotes the number of messages sent by MSS s_j that have been delivered to h_i. Initially, MH_DELIV$_i[j] = 0$ for all i, j. To reduce communication and computation loads on mobile hosts, each MH_DELIV$_i$ is stored in and maintained by MH h_i's local MSS. When a message m destined for h_j is received by h_j's local MSS, say, s_l, s_l compares the content of MH_DELIV$_j$ (which is stored at s_l) with the contents of the matrix tagging m to decide whether m is now deliverable or not. Message m is deliverable if MH_DELIV$_j[k] \geqslant m.S[k,j]$ for all $k \in \{1, \ldots, n_s\}$, where $m.S$ denotes the matrix tagging m.

In addition to vector MH_DELIV$_i$, each MSS maintains two message queues for each MH h_i resident in its cell: WAIT_ACK$_i$ and PEND$_i$. When s_l decides that a received message m is deliverable, s_l appends m to the WAIT_ACK$_i$ queue associated with destination h_i. At that time we consider m to have been delivered to h_i. MH_DELIV$_i[j]$ is then incremented by one, where j is the identifier of the MSS that sent m. Messages stored in WAIT_ACK$_i$ are sent, in sequence, to h_i over a wireless link without any loss. This property can be ensured if a link layer protocol providing orderly and reliable message delivery, such as a sliding window protocol [23], is involved. All delivered messages are thus sent to mobile hosts in the order as they are delivered. When h_i receives m, it sends back an acknowledge message, *ack*(m), to s_l. When s_l receives *ack*(m), it does the following actions in sequence. First, MSS_SENT$_l[j,i]$ is incremented by one. Second, each element in MSS_SENT$_l$ is set to the maximum of its original value and the value of the corresponding element in $m.S$. Finally, m is deleted from WAIT_ACK$_i$.

If a received message is not currently deliverable, s_l appends this message to the PEND$_i$ queue associated with h_i. Later, when some message, either a new arrival or a pending one, becomes deliverable and is delivered to h_i, the contents of MH_DELIV$_i$ are updated. This action may make one of the messages in PEND$_i$ deliverable. If so, the message is then delivered to h_i. The process continues until there is no deliverable message in PEND$_i$.

Note that unlike Alagar and Venkatesan's algorithms, MH_DELIV$_i[j]$ is incremented by one when message m, sent from MSS s_j, is appended to WAIT_ACK$_i$ at MSS s_l, while MSS_SENT$_l$ is updated after *ack*(m) sent from h_i is received by s_l. This separated updating is intended since if MSS_SENT$_l$ were updated as soon as m is appended to WAIT_ACK$_i$, m will be mistakenly considered as causally preceded any outgoing message from h_i to s_l that is received after the updating of MSS_SENT$_l$ but sent before the arrival of m at h_i. Our scheme therefore avoids a stronger but unnecessary ordering which Alagar and Venkatesan's algorithms may suffer from.

3.3. The handoff module

We assume that h_i resides in MSS s_p's cell. Suppose now h_i moves to MSS s_n's cell, which requires initiation of a handoff procedure. The first step that h_i must take is to send the message *register*(h_i, s_p) to s_n. This message informs s_n of h_i's arrival and identifies h_i's previous local MSS. In response to this message, s_n will send the message *migrate*(h_i) to s_p.

On receiving *migrate*(h_i) from s_n, s_p transmits its sending matrix, MSS_SENT$_p$, and other relevant information associated with h_i, including MH_DELIV$_i$, WAIT_ACK$_i$, and PEND$_i$, to s_n. The transmission of MSS_SENT$_p$ is to update MSS_SENT$_n$ – when MSS_SENT$_p$ is received by s_n, s_n updates MSS_SENT$_n$ as if a message had been delivered and MSS_SENT$_p$ were the sending matrix associated with the message. This action effectively maintains a monotonically-increasing sending matrix for messages originating from h_i so that their causality can be sufficiently revealed. The transmission of other data is necessary to ensure that all the messages h_i received, before or after it moved, will be delivered without violating the COMD property. After the transmission of h_i's information, if a message destined for h_i is received by s_p, it will be forwarded by s_p to s_n without modifying its contents.

The handoff procedure is completed when s_n has received all the data transmitted by s_p. From then on, s_n takes over the work of maintaining the causal ordering of messages directed to h_i. If h_i switches to another MSS, s_q, before the current handoff procedure has been completed, the *migrate* message issued by s_q will not be handled until the current handoff procedure is completed.

3.4. Handling disconnections

If h_i is disconnected from the mobile network while resident in s_l's cell, or after it leaves s_l's cell but before it enters any other cell, s_l will continue as the MSS maintaining h_i's COMD property while h_i is disconnected. All

messages destined for h_i are processed as if h_i were still connected to s_l, except that during this period, messages in WAIT_ACK$_i$ can no longer be sent to h_i. Later, when h_i is connected to the network again, messages pending at WAIT_ACK$_i$ can then be sent to h_i. If h_i has migrated to the cell of another MSS, s_n, before being reconnected, a handoff procedure between s_l and s_n must be performed before h_i can receive or send any messages.

If h_i is permanently disconnected from the system, s_l removes MH_DELIV$_i$, WAIT_ACK$_i$, and PEND$_i$ from its storage. The ith column of MSS_SENT$_l$ can also be deleted since these entries are no longer needed. This removal can be propagated to other MSSs asynchronously by piggybacking the information about h_i's disconnection on messages sent to other MSSs.

When an MH h_i that was not previously involved in computation is connected to some MSS, s_l, and applies for participation, s_l takes the following actions for h_i: First, it creates necessary data (MH_DELIV$_i$, WAIT_ACK$_i$, and PEND$_i$). Second, it inserts an associated column in MSS_SENT$_l$. This insertion can be propagated to other MSSs asynchronously in the same way as the removal is.

4. Correctness proof

Let m and m' be two messages. Let $m.S$ and $m'.S$ represent the sending matrices tagging m and m', respectively. We define two relations, $<$ and $\geqslant$, between $m.S$ and $m'.S$, as follows.

Definition 1. $m.S < m'.S$ if $m.S[i,j] \leqslant m'.S[i,j]$ for all i, j, and $\exists i', j'$, such that $m.S[i',j'] < m'.S[i',j']$.

Definition 2. $m.S \geqslant m'.S$ if $m'.S < m.S$ or $m.S[i,j] = m'.S[i,j]$ for all i, j.

The following lemmas are intermediate results necessary for our proof.

Lemma 1. If messages m and m' are sent by the same MH h_x, then *sent*(m) $\rightarrow$ *sent*(m') $\Rightarrow m.S < m'.S$.

Proof. Two cases are possible.

- Case 1: h_x does not move into any other cell after sending m and before sending m'. Let s_α be h_x's logical MSS during this period of time. Let MH h_k be m's destination. The content of $m.S$ is set to as MSS_SENT$_\alpha$ when m is sent, and after which, MSS_SENT$_\alpha[\alpha,k]$ is incremented by one. Since MSS_SENT$_\alpha$ never decreases, when m' is sent, we have $m'.S$ set to as MSS_SENT$_\alpha$ and thus $m.S < m'.S$.
- Case 2: h_x moves into other cells after sending m and before sending m'. Consider each handoff. Let h_x move from MSS s_α's cell into MSS s_β's cell. According to the handoff module, MSS_SENT$_\beta$ will be set to as MSS_SENT$_\alpha$ after handoff procedure is completed. Therefore, no matter how many handoffs are initiated during the time between the sendings of m and m', we obtain the same result as in case 1.

□

Lemma 2. If message m and m' are sent by different MHs, then *sent*(m) $\rightarrow$ *sent*(m') $\Rightarrow m.S < m'.S$.

Proof. Suppose that *sent*(m) $\rightarrow$ *sent*(m') holds, it follows that there must exists a chain of messages $m_1, m_2, \ldots, m_t$ ($t \geqslant 2$) such that m and m_1 are sent from the same MH (m_1 might be m), $m_t \equiv m'$, and m_i ($1 \leqslant i \leqslant t-1$) is delivered to the same MH as m_{i+1} is sent from. Consider each message m_i. Suppose that m_i is sent from MH h_x and is delivered to MH h_y. Let s_α be h_x's local MSS when m_i is sent and let s_β be h_y's local MSS when m_i is delivered. From our protocol,

$$m_i.S = \text{MSS\_SENT}_\alpha \tag{1}$$

when m_i is sent, and

$$m_i.S < \text{MSS\_SENT}_\beta \tag{2}$$

after m_i has been delivered (since after the delivery of m_i, MSS_SENT$_\beta[\alpha, y]$ is incremented by one and each other element in MSS_SENT$_\beta$ is set to the maximum of its original value and the value of the corresponding element in $m_i.S$). If h_y does not move outside s_β's cell before sending m_{i+1}, we have

$$m_{i+1}.S \geqslant \text{MSS\_SENT}_\beta; \tag{3}$$

otherwise, if h_y migrates to other cells before sending m_{i+1}, our handoff procedure ensures that MSS_SENT matrix in the new MSS is never smaller than that in the old MSS. Hence we obtain the same result as in (3). It follows from (1)–(3) that $m_i.S[\ ] < m_{i+1}.S[\ ]$ for all i, $1 \leqslant i \leqslant t-1$. By transitivity, we have $m.S[\ ] < m'.S[\ ]$. □

Lemma 3. Suppose n messages (n is finite) have been sent by MSS s_α and are destined for MH h_x. Let m_i ($1 \leqslant i \leqslant n$) denote the ith message. If m_i has not been delivered to h_x, MH_DELIV$_x[\alpha] \leqslant m_i.S[\alpha, x]$.

Proof. By our protocol, $m_i.S[\alpha, x] = i - 1$. By way of contradiction, suppose that m_i has not been delivered but MH_DELIV$_x[\alpha] > m_i.S[\alpha, x] = i - 1$. Since MH_DELIV$_x[\alpha]$ is incremented by one only when a message sent by s_α is delivered to h_x, MH_DELIV$_x[\alpha] > i-1$ implies that more than $i-1$ messages have been delivered to h_x, therefore, some message m_j, where $j > i$, must have been delivered. According to our algorithm, the premise to deliver m_j is that MH_DELIV$_x[\alpha] \geqslant m_j.S[\alpha, x] = j - 1$, which implies that at least $j-1$ messages must have been delivered before the delivery of m_j. Since at least one message that was sent before m_j has not been delivered (i.e., h_i), some message m_k ($k > j$) must have

been delivered before the delivery of m_j. The same argument applied to m_j can apply to m_k, and so on. Since n is finite, eventually we will end up with the condition MH_DELIV$_x[\alpha] > n$, which is impossible and therefore a contradiction. □

The following theorem shows the correctness of our algorithm.

Theorem 1. Let m and m' be two messages. *sent*(m) → *sent*(m') ⇒ *deliv*(m) → *deliv*(m').

Proof. Let m be sent from MH h_x and m' be sent from MH h_y. Let s_α be h_x's local MSS when m is sent and s_β be h_y's local MSS when m' is sent. Let m and m' be both destined for MH h_z. Suppose that m' has been delivered but m has not, we have MH_DELIV$_k[\alpha] \geqslant m'.S[\alpha, z]$. Since $m.S[\alpha, z] < m'.S[\alpha, z]$ (from lemmas 1 and 2), it follows that MH_DELIV$_k[\alpha] > m.S[\alpha, z]$. Therefore, from lemma 3, m must have been delivered, which is a contradiction. □

5. Comparisons

We compare our algorithm with the three counterparts proposed by Alagar and Venkatesan along three dimensions: message overhead, handoff cost, and the probability of unnecessary inhibition in delivering messages. For abbreviation, we denote our algorithm and the first, the second, and the third algorithms as YHH, AV94-1, AV94-2, and AV94-3, respectively.

5.1. Message overhead

All these algorithms need to add extra bits to the header of each message. These bits are essentially overhead that will increase message transmission delay when the messages pass along the wired network. However, the amount of overhead each of these algorithms adds to messages varies. The needed header sizes per message are O(n_h^2) in AV94-1, O(n_s^2) in AV94-2, and O($k^2\, n_s^2$) in AV94-3, where k is a predetermined integer parameter. Our algorithm requires a header size of O($n_s \times n_h$) in each message. Let $p = n_h/n_s$. Typically, p is greater than one, in which case AV94-2 has the lowest message overhead. The message overheads of the others, when p is greater than one, are shown in table 1. In this table it can be seen that our algorithm outperforms the other two with respect to message overhead when $p < k^2$.

5.2. Handoff cost

The handoff cost can be measured by the length of time a handoff procedure takes. This measure is directly proportional to the amount of information transmitted by the handoff procedure. The kind of information sent by AV94-1 is the same as that sent by YHH: the SEND matrix, the DELIV vector, and the WAIT_ACK and PEND message queues. However, the sizes of matrix SEND and vector DELIV are O($n_s \times n_h$) and O(n_s), respectively, in the YHH algorithm, while AV94-1 needs O(n_h^2) and O(n_h) to transmit the same information. AV94-2 and AV94-3 do not send information associated with the migrating MH in the handoff procedure. They use, however, O($k\, n_s$) control messages, each of which has a header size of O($k^2\, n_s^2$) ($k = 1$ in AV94-2). Therefore, the handoff costs are O(n_h^2) + O(q) in AV94-1, O(n_s) × O(n_s^2) in AV94-2, O($k\, n_s$) × O($k^2\, n_s^2$) in AV94-3, and O($n_s \times n_h$) + O(q) in YHH, where q represents the number of messages pending in the message queues when the handoff procedure begins. We assume that O(q) < O($n_s \times n_h$). The comparisons of the handoff costs, with various ranges of p, are shown in table 2. In this table it is easy to see that our algorithm has the lowest handoff cost of all when $p \leqslant n_s$.

There is another factor that makes the handoff costs even higher in AV94-2 and AV94-3. In these two algorithms, all MSSs (more precisely, all $k \times n_s$ MSSs) are involved in the handoff procedure. The handoff procedure can not be completed until all MSSs, in response to a *notify* message broadcast by a certain MSS, have sent back their acknowledgments, and all the acknowledgments have been received by the MSS that issued the original *notify* message. All the *notify* messages and the associated acknowledgments must be causally ordered, meaning that extra delays in delivering these messages may be needed to maintain their causality. This penalty is greater than expected when the possibility of unnecessary inhibition is also considered.

Table 1
Comparison of message overheads among YHH, AV94-1, and AV94-3.

The range of p	Order of overheads
$1 < p \leqslant k$	YHH < AV94-1 ⩽ AV94-3
$k < p \leqslant k^2$	YHH ⩽ AV94-3 < AV94-1
$k^2 < p$	AV94-3 < YHH < AV94-1

Table 2
Comparison of handoff costs among all algorithms.

(a) When $n_s > k^3$

The range of p	Order of costs
$1 < p \leqslant \sqrt{n_s}$	YHH < AV94-1 ⩽ AV94-2 < AV94-3
$\sqrt{n_s} < p \leqslant k^{3/2}\sqrt{n_s}$	YHH < AV94-2 < AV94-1 ⩽ AV94-3
$k^{3/2}\sqrt{n_s} < p \leqslant n_s$	YHH ⩽ AV94-2 < AV94-3 < AV94-1
$n_s < p \leqslant k^3 n_s$	AV94-2 < YHH ⩽ AV94-3 < AV94-1
$k^3 n_s < p$	AV94-2 < AV94-3 < YHH < AV94-1

(b) When $n_s \leqslant k^3$

The range of p	Order of costs
$1 < p \leqslant \sqrt{n_s}$	YHH < AV94-1 ⩽ AV94-2 < AV94-3
$\sqrt{n_s} < p \leqslant n_s$	YHH ⩽ AV94-2 < AV94-1 ⩽ AV94-3
$n_s < p \leqslant k^{3/2}\sqrt{n_s}$	AV94-2 < YHH < AV94-1 ⩽ AV94-3
$k^{3/2}\sqrt{n_s} < p \leqslant k^3 n_s$	AV94-2 < YHH ⩽ AV94-3 < AV94-1
$k^3 n_s < p$	AV94-2 < AV94-3 < YHH < AV94-1

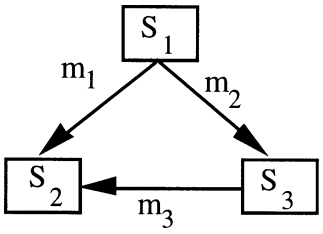


Figure 2. An example of unnecessary inhibition.

5.3. Unnecessary delivery inhibition

Before we compare the probability of unnecessary delivery inhibition among all algorithms, we illustrate the phenomenon of unnecessary inhibition with an example. Consider the system configuration shown in figure 2. Suppose that MSS s_1 sends message m_1 to MSS s_2 and then sends message m_2 to MSS s_3. After delivering m_2, s_3 sends message m_3 to s_2. If m_3 is received by s_2 before m_1 arrives, AV94-2 inhibits delivery of m_3 until m_1 has been delivered, since, with respect to the MSSs' point of view, *send*(m_1) $\rightarrow$*send*(m_3) holds and thus, m_3 should not be delivered before the delivery of m_1. However, this inhibition is only necessary when *send*(m_1) $\rightarrow$*send*(m_3) also holds at the MH level. In this example, the necessary conditions are as follows:

- Messages m_1 and m_2 originate from the same MH.
- Message m_3 originates from the same MH m_2 is destined for.
- Messages m_3 and m_1 are destined for the same MH.

If these conditions do not all hold simultaneously, the inhibition is unnecessary.

Unnecessary inhibition does not exist at all in AV94-1. On the other hand, AV94-2 has the highest probability of unnecessary inhibition since it only maintains the COMD property at the MSS level. AV94-3 tries to reduce the probability of unnecessary inhibition by allocating MHs in cells to different logical MSSs, and maintaining the COMD property among logical MSSs. Consequently, messages to MHs belonging to different logical MSSs will not inhibit each other even though these MHs may be in the same cell. Our algorithm also deals with the problem of unnecessary inhibition by confining the enforcement of the COMD property to messages that are destined for the same MH. As in AV94-2, a message sent by an MSS is considered to be causally related to all messages that have been sent or delivered by the same MSS. However, only messages directed to the same MH will be considered for inhibition to maintain COMD. Messages to different MHs will not inhibit each other even though they may be directed to the same MSS. Hence the probability of unnecessary inhibition can be reduced.

To compare the probability of inhibition in AV94-3 with that in YHH, consider the example shown in figure 2 again. Let each MSS be partitioned into k logical MSSs, where $k \geqslant 2$. Suppose that the following conditions hold:

- The originators of messages issued from a cell and the destinations of messages delivered to a cell are both uniformly distributed over all MHs in that cell.
- MHs are uniformly distributed over all cells.
- MHs in a cell are uniformly distributed over all logical MSSs.

In AV94-3, when m_3 arrives at s_2 before m_1 arrives, the delivery of m_3 will be temporarily inhibited only if all the following conditions hold:

(C1) Messages m_1 and m_2 were sent by the same logical MSS.

(C2) Message m_3 was sent by the same logical MSS m_2 is delivered to.

(C3) Messages m_3 and m_1 are directed to the same logical MSS.

It is easy to see that the probabilities of (C1), (C2) and (C3) are all $1/k$. So, the probability of inhibition will be $1/k^3$ of that in AV94-2. In our approach, when m_3 arrives at s_2 before m_1 arrives, the delivery of m_3 is temporarily inhibited only if m_3 and m_1 are destined for the same MH. The probability of inhibition will be $1/p$ of that in AV94-2, where $p = n_h/n_s$ is the expected number of MHs resident in a cell. This degree of probability will be lower than AV94-3 when $p > k^3$, and higher than that otherwise.

Both AV94-3 and our algorithm reduce the probability of unnecessary inhibition inherent in AV94-2. The performance of AV94-3 depends on the value of k. As k increases, unnecessary delays in delivering messages to MHs decrease. However, as k increases message overheads and handoff costs also increase. So the optimal setting of k must be evaluated by simulation or experiment. In contrast, our approach provides an alternative whose behavior is more deterministic.

6. Conclusions

We propose an algorithm that provides causally ordered message delivery for mobile computing systems. This algorithm is mainly executed by MSSs, and thus reduces both the amount of computation performed by mobile hosts and the communication overhead in wireless channels. Previous solutions of the same problem either incur high message overheads or suffer from high hand-off costs. Our approach makes a compromise between these two extremes: the hand-off cost is low while the message overhead is median. In addition, our approach reduces the probability of unnecessary inhibition in delivering messages that conventional low-message-overhead approaches incur.

Prakash et al. [18] have proposed another causal ordering algorithm for mobile computing environment. In their algorithm, each message needs to carry information only about its direct predecessor message, with respect to each destination MH, and most recent mutually concurrent message delivered to its sender. This strategy usually can reduce message overhead. However, the worst-case message

overhead is still O(n_h^2). Moreover, since this algorithm is implemented on each MH, it suffers from the same disadvantages as AV94-1 does. Nevertheless, we think that if the strategy applied in this algorithm is adapted and integrated into our approach, the message overhead of our algorithm might be further reduced.

References

[1] A. Acharya and B.R. Badrinath, Delivering multicast messages in networks with mobile hosts, in: *Proc. 13th International Conference on Distributed Computing Systems* (May 1993).

[2] A. Acharya and B.R. Badrinath, Checkpointing distributed applications on mobile computers, in: *Proc. 3rd International Conference on Parallel and Distributed Information Systems* (September 1994).

[3] S. Alagar and S. Venkatesan, Causally ordered message delivery in mobile systems, in: *Proc. Workshop on Mobile Computing Systems and Applications* (December 1994) pp. 169–174.

[4] L. Alvisi and K. Marzullo, Message logging: Pessimistic, optimistic, and causal, in: *Proc. 15th International Conference on Distributed Computing Systems* (May 1995).

[5] V. Aravamudhan, K. Ratnam and S. Rangarajan, An efficient multicast protocol for PCS networks, in: *Proc. 2nd International Mobile Computing Conference*, Hsinchu, Taiwan (March 1996) pp. 25–34.

[6] B.R. Badrinath, A. Acharya and T. Imielinski, Impact of mobility on distributed computations, Oper. Syst. Rev. 27(2) (April 1993) 15–20.

[7] B.R. Badrinath, A. Acharya and T. Imielinski, Structuring distributed algorithms for mobile hosts, in: *Proc. 14th International Conference on Distributed Computing Systems* (June 1994).

[8] K.P. Birman and T.A. Joseph, Reliable communications in the presence of failures, ACM Trans. Comput. Syst. 5(1) (February 1987) 47–76.

[9] K.P. Birman and T.A. Joseph, Exploiting replication in distributed systems, in: *Distributed Systems*, ed. S. Mullender (Addison-Wesley, New York, 1989).

[10] G.H. Forman and J. Zahorjan, The challenges of mobile computing, IEEE Computers 27(4) (April 1994) 38–47.

[11] J. Ioannidis, D. Duchamp and G.Q. Maruire Jr., IP-based protocols for mobile internetworking, in: *Proc. ACM SIGCOMM Symposium on Communication Architecture and Protocols* (1991) pp. 235–245.

[12] L. Lamport, Time, clocks, and the ordering of events in a distributed system, Comm. ACM 21(7) (July 1978) 538–565.

[13] V.O.K. Li and X. Qiu, Personal communication systems (PCS), Proceedings of the IEEE 83(9) (September 1995) 1210–1243.

[14] F. Mattern, Virtual time and global states of distributed systems, in: *Proc. International Workshop on Parallel and Distributed Algorithms*, eds. M. Cosnard, Y. Robert, P. Quinton and M. Raynal (North-Holland, Elsevier Science, 1989) pp. 215–226.

[15] S. Mishra, L. Peterson and R. Schlicting, Implementing fault-tolerant replicated objects using Psync, in: *Proc. 14th Symposium on Operating System Principles*, Asheville, North Carolina (December 1993).

[16] A. Mostefaoui and M. Raynal, Causal multicasts in overlapping groups: towards a low cost approach, in: *Proc. 4th Workshop on Future Trends of Distributed Computing Systems*, Lisbon, Portugal (September 1993) pp. 136–142.

[17] C. Perkins, A. Myles and D.B. Johnson, IMHP: a mobile host protocol for the Internet, Computer Networks and ISDN Systems 27 (1994) 479–491.

[18] R. Prakash, M. Raynal and M. Singhal, An efficient causal ordering algorithm for mobile computing environments, Technical Report OSU-CISRC-TR44, Department of Computer and Information Science, Ohio State University (1995).

[19] R. Prakash and M. Singhal, Low-cost checkpointing and failure recovery in mobile computing systems, Technical Report OSU-CISRC-6/94-TR36, Department of Computer and Information Science, Ohio State University (1994).

[20] M. Raynal, A. Schiper and S. Toueg, The causal ordering abstraction and a simple way to implement it, Inform. Process. Lett. 39 (September 1991) 343–350.

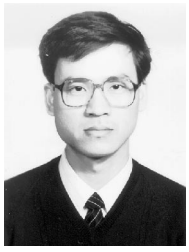
[21] L.E. Rodrigues and P. Verissimo, Causal separators for large-scale multicast communication, in: *Proc. 15th International Conference on Distributed Computing Systems* (June 1995) pp. 83–91.

[22] A. Schiper, J. Eggli and A. Sandoz, A new algorithm to implement causal ordering, in: *Proc. 3rd International Workshop on Distributed Algorithms*, 1989. Also published in Lecture Notes in Computer Science 392.

[23] A.S. Tanenbaum, *Computer Networks* (Prentice-Hall, Englewood Cliffs, NJ, 2nd ed., 1988) chapter 4, pp. 223–239.

Li-Hsing Yen received the B.S. and M.S. degrees in computer science and information engineering, both from National Chiao Tung University, Hsinchu, Taiwan, in 1989 and 1991, respectively. Since September 1993, he has been a Ph.D. student in the Department of Computer Science and Information Engineering at National Chiao Tung University, Hsinchu, Taiwan. His current research interests include distributed algorithms, program testing and verification, and mobile computing.

E-mail: lsyen@csie.nctu.edu.tw



Ting-Lu Huang studied at Tung-hi University (B.S., 1976), University of Texas at Arlington (M.S., 1981), and Northwestern University (Ph.D., 1989). He is currently an associate professor in the Department of Computer Science and Information Engineering at National Chiao Tung University. He has been working on mutual exclusion algorithms, causality in distributed computing, testing and debugging of concurrent software, and verification. Dr. Huang is a member of the ACM.

Shu-Yuen Hwang is a Professor in the Department of Computer Science and Information Engineering, National Chiao Tung University. He received the B.S. and M.S. degrees in electrical engineering from National Taiwan University in 1981 and 1983; and the Ph.D. degree in computer science from the University of Washington in 1989. His current research interests include artificial intelligence, computer simulation and mobile computing.