Causal Ordering in Distributed Mobile Systems

Sridhar Alagar and S. Venkatesan, *Member*, *IEEE*

Abstract—There is a growing trend in using mobile computing environment for several applications, and it is important that the mobile systems are provided adequate support both at the systems level and at the communication level. *Causal ordering* is a useful property, particularly in applications that involve human interactions. (Such applications are common in mobile computing environments.) In this paper, we present three algorithms for causal ordering in mobile sytems. The first algorithm handles the resource constraints of the mobile hosts, but the system is not easily scalable with respect to the number of mobile hosts and is not graceful to host disconnections and connections. Our second algorithm eliminates the above disadvantages at the cost of inhibiting some messages. The third algorithm is a combination of the first two algorithms.

Index Terms—Mobile computing, asynchrony, causal ordering, message overhead, communication complexity.

1 Introduction

The emergence of PDAs, PCS, wireless LAN, etc., has made mobile computing widely realizable in practice. The mobility of hosts poses various challenges and problems in designing distributed algorithms. Many of the distributed algorithms designed for static hosts cannot be directly used for mobile systems due to the change in physical connectivity, resource constraints of mobile hosts, and limited bandwidth of the wireless links [5]. This has spawned a considerable amount of research in mobile computing: from designing communication protocols [1], [8], [18] to providing fault tolerance [3], [10]. In this paper, we consider the problem of providing a particular kind of communication support, namely, *causally ordered* message delivery to mobile hosts.

One of the fundamental problems in distributed computing is controlling the nondeterminism. Nondeterminism arises due to the asynchronous nature of the hosts and the communication medium [15]. In order to reduce the nondeterminism in the communication medium, Birman and Joseph [7] first proposed causal ordering for message delivery. Consider two messages m and m' sent to the same destination such that sending of m "happened before" sending of m'. Causal ordering is obeyed if m is received before m' is received.

Causal ordering is very useful in several applications like management of replicated data, resource allocation, monitoring a distributed system, USENET, etc. [2], [6], [15], [19]. Causal ordering is best suited for applications that involve human interactions from several locations [19], which are typical in mobile systems. Some of the major applications of distributed mobile systems, at present and in future, include providing information, stock trading, teleconferencing, etc., it is undesirable for a user to receive messages out of causal order.

• *The authors are with the Department of Computer Science, University of Texas at Dallas, Richardson, TX 75083.*
E-mail: {sridhar, venky}@utdallas.edu.

For information on obtaining reprints of this article, please send e-mail to: transcom@computer.org, and reference IEEECS Log Number C97025.

1.1 Motivation

Any of the existing algorithms for causal ordering in static hosts can be executed by every mobile host, and all the relevant data structures can be stored in the mobile hosts. But while designing algorithms for mobile systems, the following factors must be taken into account.

F1. The capability of mobile hosts may be very limited in terms of the disk space and processing power [4]. The amount of computation performed by a mobile host should be low.

F2. The available bandwidth is low and the cost of message transmission is high in the wireless medium compared to the wired medium [13]. So the communication overhead in the wireless medium should be minimal.

F3. The number of mobile hosts (MHs) may not be known in advance and it may be substantially larger than the number of mobile support stations (MSSs). Hence it is desirable to design algorithms such that the overhead (communication and computation) does not increase with the number of mobile hosts.

F4. A mobile host may often disconnect itself from the network and reconnect at a later time. Algorithms designed for mobile systems should be able to easily handle the effect of host disconnections and connections.

If mobile hosts are made to execute the traditional causal ordering algorithms (by storing the relevant data structures in the mobile hosts), none of the above factors (F1–F4) can be satisfied. To keep the computation performed by mobile hosts and the cost of wireless communication low, MSSs can execute some tasks on behalf of the MHs. We present three algorithms for causal ordering in mobile systems.

In the first algorithm, MSSs store the relevant data structures of the MHs and execute the causal ordering algorithm on behalf of the MHs in their cells. The algorithm satifies factors F1 and F2. The message overhead (length of the header[1]) is proportional to the square of the number of mobile hosts. Thus, factor F3 is not satisfied. Also, the

1. *Header* of a message is the extra information sent with the message to maintain causal ordering.

TABLE 1
NH IS THE NUMBER OF MHS, NS IS THE NUMBER OF MSSS, AND K IS THE NUMBER OF LOGICAL MSSS PER MSS

Algorithm	Size of message header	"handoff" Complexity	"inhibition"
Algorithm 1	$O(n_h^2)$ integers	$O(1)$ messages	No
Algorithm 2	$O(n_s^2)$ integers	$O(n_s)$ messages	Yes
Algorithm 3	$O(k^2 * n_s^2)$ integers	$O(k * n_s)$ messages	decreases as k increases

$n_h \gg n_s, k \geq 1$

algorithm does not handle hosts disconnections and connections (factor F4) very well and is not easily scalable.

Algorithm 2 eliminates the problems in Algorithm 1. The size of the message header is proportional to the square of the number of MSSs. Since the size of the header does not vary with the number of mobile hosts, the algorithm is scalable (with respect to the number of the mobile hosts) and host disconnections/connections do not pose any problem. But there may be some "inhibition" in delivering the messages to the mobile hosts. Our experimental results suggest that delay due to inhibition is less than the delay involved in transmitting and processing the long header (of each message) used in Algorithm 1. Also, the load placed on the MSSs is less than that of Algorithm 1, because an MSS need not maintain the data structures on behalf of each mobile host in its cell.

Algorithm 3 is a hybrid algorithm and exhibits a tradeoff between Algorithm 1 and Algorithm 2. Every MSS is partioned into k logical MSSs to reduce the delay due to "inhibition" in delivering the messages to MHs. However, k cannot be large as this will increase the message overhead. A summary of our analytical results is shown in Table 1.

2 MODEL AND DEFINITIONS

A distributed mobile system consists of a set of *mobile hosts* and *static hosts*. A *mobile host* (MH) is a host whose geographical location can change with time while retaining its connectivity to the network [8]. A *static host* is a host whose physical location does not change. A static host can also be a *mobile support station* (MSS). An MSS has the necessary infrastructures to support and communicate with the MHs. For simplicity, we assume that the system consists of only MSSs and MHs. A static host can be considered as an MH that does not move.

An MSS communicates with the MHs through a wireless channel. The geographical area within which an MSS communicates with MHs is called a *cell*. An MH can communicate directly with an MSS only if the MH is located in the cell of the MSS. A mobile host may belong to at most one cell at any time. Mobile hosts communicate with other hosts through their MSSs.[2] MSSs are connected among themselves using wired channels. The MSSs and the wired channels constitute the static network. We assume that a *logical* channel exists between every pair of MSSs. These logical channels need not be FIFO channels, whereas the wireless channels obey the FIFO property. Both wired and wireless channels are reliable and take an arbitrary, but finite amount of time to deliver messages. (Though a physical wireless medium does not take an arbitrary amount of time, the end-to-end delay for a message from an MH to its MSS could be arbitrary due to message retransmission and queuing delays.) A mobile host can migrate from one cell to another cell at any time.

EXAMPLE. An example of a distributed mobile system is shown in Fig. 1. s_1, s_2, and s_3 are three MSSs connected by wired channels. h_1, h_2, and h_3 are three mobile hosts. Initially, h_1 and h_3 are in the cell of s_1, and h_2 is in the cell of s_3. Let MH h_1 move from the cell of MSS s_1 to the cell of s_2 as shown in Fig. 1. h_1 discovers that it is in the cell of s_2 after receiving the *beacon* [8] broadcast by s_2. MH h_1 informs MSS s_2 of h_1's *id* and the *id* of its previous MSS s_1. A *handoff* procedure is then executed between s_2 and s_1. MSS s_2 informs s_1 about h_1's migration and receives the relevant information (associated with h_1) from MSS s_1.

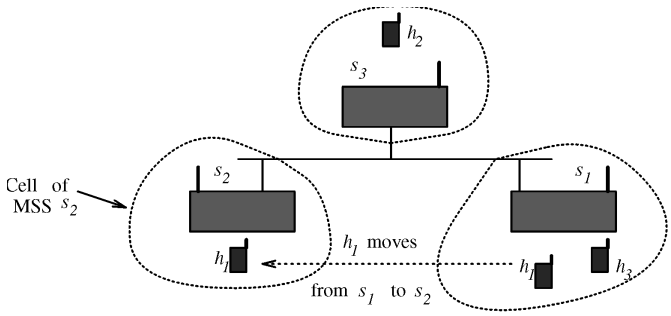


Fig. 1. Mobile system.

A mobile host can disconnect itself from the network by sending a *disconnect* message to its current MSS and can reconnect at a later time by sending a *connect* message. If an MSS receives a message destined for any of the disconnected mobile hosts, the message can be stored and delivered to mobile host after it reconnects, or the message can be discarded, depending on the application.

An event in a host may be a send event (sending a message to another host), a receive event (receiving a message from a host), or an internal event which does not involve communication. Let *send*(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. Events are ordered by Lamport's "happened before" relation $\rightarrow$ [12]. For any two events e and e', $e \rightarrow e'$ is true if

1) e and e' are two events in the same host and e occurs before e' or

2. The MSS of an MH is the MSS in whose cell the MH is located.

2) e corresponds to sending a message m and e' corresponds to the receipt of m or
3) there exists an event e'' such that $e \rightarrow e''$ and $e'' \rightarrow e'$.

Causal ordering of message delivery is obeyed if, for any two messages m and m' that have the same destination, $send(m) \rightarrow send(m')$ implies that $recv(m) \rightarrow recv(m')$.

EXAMPLE. Consider the example shown in Fig. 2. Messages m_1 and m_3 are sent to host P_3, and $send(m_1) \rightarrow send(m_3)$. If P_3 receives m_3 first, causal ordering will be violated. Causal ordering will be respected if P_3 receives m_1 and then m_3. Let m_1 and m_2 be "requests for an information" and m_3 be a reply to the request sent by P_1. If m_3 is received by P_3 before receiving m_1, P_3 is "puzzled." At P_3, the requested information appears before the request.

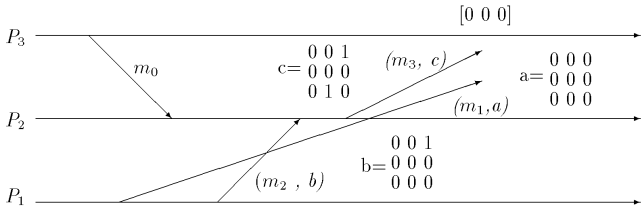


Fig. 2. A sample execution.

3 PRELIMINARIES

Causal ordering was first proposed for the ISIS system [7]. There are several algorithms that implement causal ordering for distributed systems with static hosts [7], [15], [16]. The algorithm by Birman and Joseph appends, to every message, the history of the communications that happened before the sending of the message [7]. The size of the message header may be unbounded but the channels need not be reliable. The algorithm by Raynal, Schiper, and Toueg, referred to henceforth as the RST algorithm, is based on message counting and assumes the channels to be reliable [15]. The RST algorithm, which is discussed subsequently, appends N^2 integers to every message, where N is the number of hosts in the system. The algorithm by Schiper et al. [16] uses vector clocks and is somewhat similar to the RST algorithm.

3.1 RST Algorithm

The RST algorithm for causal ordering maintains two arrays, $\text{DELIV}_i[N]$ and $\text{SENT}_i[N,N]$, for each host P_i. $\text{DELIV}_i[j]$ denotes the total number of messages received by P_i from P_j. $\text{SENT}_i[k, j]$ indicates P_i's knowledge about the number of messages P_k has sent to P_j. The following steps are executed at P_i to ensure causal ordering.

Whenever P_i sends message M to P_j, P_i appends its current SENT_i matrix to M. (P_i sends (M, SENT_i) to P_j.) Observe that SENT_i contains information about the messages that were sent before M was sent. P_i then increments $\text{SENT}_i[i, j]$ by 1.

On receiving a message containing (M, ST) from P_j, the causal ordering algorithm at P_i first checks if $\text{DELIV}_i[k] \geq ST[k, i]$ for $1 \leq k \leq N$. (P_i checks whether all the messages, sent to P_i, that are causally dependent on M have been delivered.) If so, message M is delivered to the application, $\text{DELIV}_i[j]$ is incremented by 1, $\text{SENT}_i[j, i]$ is set to $ST[j, i] + 1$, and, finally, $\text{SENT}_i[i, j]$ is set to $max(ST[i, j], \text{SENT}_i[i, j])$ for all i, j. If not, M is buffered till $\text{DELIV}_i[k] \geq ST[k, i]$ for $1 \leq k \leq N$.

EXAMPLE. Consider the sample execution consisting of three hosts shown in Fig. 2. Host P_1 sends message (m_1, SENT_1) to P_3 and increments $\text{SENT}_1[1, 3]$ by 1. P_1 then sends message (m_2, SENT_1) to P_2. On receiving (m_2, ST_{m_2}), P_2 updates SENT_2 based on ST_{m_2}. After updating SENT_2, P_2 gains the knowledge that P_1 has sent one message to P_3 before sending m_2 to P_2 (the value of $ST_{m_2}[1, 3]$ is 1). P_2 transfers this knowledge to P_3 when it sends (m_3, SENT_3) to P_3. Assume that m_3 arrives at P_3 before m_1. Now m_3 will not be delivered to the application, because $\text{DELIV}_3[1] \not\geq ST_{m_3}[1, 3]$. Message m_3 will be delivered only after m_1 is delivered.

3.2 Reliable Message Delivery

Before we explain our algorithms, we describe a simple protocol to ensure reliable message delivery (without duplication) when the MH moves from one cell to another cell.

Let MH h_i be in the cell of MSS s_j. Every message generated by h_i is first sent to s_j and then s_j sends the message to the destination. Similarly, every message received by h_i is received through MSS s_j. Let MH_RSEQNO_i denote the number of messages received by MH h_i from MSS s_j. Let MH_SSEQNO_i denote the number of messages sent by h_i that have been received by s_j. MH_RSEQNO_i is maintained at MH h_i and MH_SSEQNO_i is maintained at the MSS of MH h_i. Messages sent by s_j to h_i are numbered sequentially in increasing order and are stored in a FIFO queue, PEND_ACK_i, within s_j. MH h_i sends an ack after receiving a message from s_j and increments MH_RSEQNO_i. After receiving an ack for a message it sent to h_i, MSS s_j deletes that message from PEND_ACK_i. Messages sent by h_i to s_j are also sequentially numbered by h_i sequentially in the increasing order and are stored locally in MH h_i till the messages are acknowledged. After receiving a message from h_i, MSS s_j sends an ack to h_i and increments MH_SSEQNO_i.

Next the algorithms for causal ordering are described.

4 ALGORITHM 1

In this section, the RST algorithm is extended to mobile systems. Algorithm 1 consists of two modules: static module and handoff module. The static module is executed when an MH is in a particular cell. The handoff module is executed when an MH moves from one cell to another.

4.1 Static Module

For each MH h_i, we maintain two arrays $\text{MH\_DELIV}_i[n_h]$ and $\text{MH\_SENT}_i[n_h,n_h]$, where n_h is the number of mobile hosts in the mobile computing environment. $\text{MH\_DELIV}_i[j]$ denotes the total number of messages received by h_i from h_j. $\text{MH\_SENT}_i[k, j]$ indicates h_i's knowledge about the number of messages h_k has sent to h_j. Assume that MH h_i is in the cell

of MSS s_j. To reduce the communication and computation overhead of MH h_i, these arrays are stored in MSS s_j. Since the messages from (to) h_i go through MSS s_j, the causal ordering algorithm for MH h_i is executed by MSS s_j. A message M sent to MH h_i becomes *deliverable* to h_i if the reception of M by h_i does not violate causal ordering. All messages for h_i, received by s_j, which are not yet deliverable are stored in the queue MH_PENDING$_i$ at MSS s_j.

Initially, all the entries in MH_DELIV$_i$ and MH_SENT$_i$ are set to 0 and MH_PENDING$_i$ is empty. To send a message m to another MH h_j, h_i first sends the message m to its MSS s_j. s_j tags MH_SENT$_i$ with m, and sends (m, MH_SENT$_i$) to the MSS of h_j. There are several protocols [8], [9], [18] that ensure message delivery to mobile hosts. Any of these protocols can be used.

MSS s_j, on receiving a message (m, ST_m) meant for h_i from MH h_j, first checks whether m is deliverable. Message m is deliverable if MH_DELIV$_i[k] \geq ST_m[k, i]$ for all k. If m is not deliverable, m is stored in MH_PENDING$_i$ till m becomes deliverable. If m is deliverable, s_j transmits m to h_i, increments MH_DELIV$_i[j]$. MSS s_j then queues message (m, ST_m) in PEND_ACK$_i$. After receiving an ack for m from MH h_i, MSS s_j updates array MH_SENT$_i$ and deletes message (m, ST_m) from PEND_ACK$_i$. Whenever a message m is delivered to h_i, s_j checks MH_PENDING$_i$ for any message that may be deliverable after m was delivered. A formal description of the module is given in Fig. 3.

Let MH h_i be in the cell of MSS s_j.

1. On receiving message m from h_i to be sent to MH h_j, MSS s_j executes the following steps.
   (a) Send (m, MH_SENT$_i$) to h_j.
   (b) MH_SENT$_i[i, j]$ = MH_SENT$_i[i, j] + 1$.
   (c) MH_SSEQNO$_i$ = MH_SSEQNO$_i$ + 1.
   (d) Send an ack to h_i.

2. MSS s_j, on receiving a message m for h_i from h_j, executes the following steps.
   (a) If m is not deliverable to h_i, queue (m,ST_m) in MH_PENDING$_i$.
   (b) If m is deliverable, then
      i. Transmit m to h_i and queue (m,ST_m) in PEND_ACK$_i$.
      ii. MH_DELIV$_i[j]$ = MH_DELIV$_i[j] + 1$.
      iii. MH_SENT$_i[j, i] = ST_m[j, i] + 1$.
      iv. After receiving an ack for m from h_i do the following. set MH_SENT$_i[k, l]$ = *max*(MH_SENT$_i[k, l]$, $ST_m[k, l]$), for all k, l. Delete (m, ST_m) from PEND_ACK$_i$.
      v. If any message (m, ST_m) in MH_PENDING$_i$ becomes deliverable, goto step 2(b)i.

Fig. 3. Static module of Algorithm 1.

4.2 Handoff Module

Let h_i move from the cell of MSS s_j to the cell of MSS s_k. The handoff module is then executed by s_j and s_k. After entering the cell of s_k, MH h_i sends the message *register*(h_i, s_j, MH_RSEQNO$_i$) to s_k. Also, h_i retransmits the messages (to s_k) for which it did not receive ack from its previous MSS s_j. MSS s_k then informs s_j that h_i has switched from MSS s_j to MSS s_k by sending a *handoff_begin*(h_i) message to s_j. After receiving *handoff_begin*(h_i), s_j transfers MH_DELIV$_i$, MH_SENT$_i$, MH_PENDING$_i$, PEND_ACK$_i$, and MH_SSEQNO$_i$ to MSS s_k and finally sends message *handoff_over*(h_i) to s_k. After sending the *handoff_over* message, if s_j receives any messages sent by h_i (when h_i was in s_j's cell), MSS s_j drops them. (This scenario is possible because the wireless channels take an arbitrary amount of time to deliver messages.)

On receiving these data structures, s_k first transmits all messages in PEND_ACK$_i$ with sequence number greater than MH_RSEQNO$_i$ in FIFO order. Also, s_k forwards the messages (to their destinations) retransmitted by h_i with sequence number greater than MH_SSEQNO$_i$. The handoff procedure is then terminated at s_k. A description of the handoff module appears in Fig. 4. If MH h_i switches to some other cell before the handoff is completed, the current handoff is completed before a new handoff begins.

Let MH h_i switch from the cell of MSS s_j to the cell of MSS s_k.

1. Steps executed by MSS s_k.
   (a) On receiving message *register*(h_i, s_j, MH_RSEQNO$_i$) from h_i, send message *handoff_begin*(h_i) to MSS s_j.
   (b) Receive MH_SENT$_i$, MH_PENDING$_i$, PEND_ACK$_i$, MH_SSEQNO$_i$, and *handoff_over*(h_i) from s_j.
   (c) For each message (m, ST_m) in PEND_ACK$_i$ with number less than MH_RSEQNO$_i$, set MH_SENT$_i[k, l]$ = *max*(MH_SENT$_i[k, l]$, $ST_m[k, l]$), for all k, l and delete the message from PEND_ACK$_i$.
   (d) For each message (m, ST_m) in PEND_ACK$_i$ with number greater than MH_RSEQNO$_i$, transmit m to h_i in FIFO order. Update MH_SENT$_i$ with ST_m and delete the message from PEND_ACK$_i$ after receiving an ack for m from h_i.
   (e) Send messages sent by h_i with number greater than MH_SSEQNO$_i$ to their destinations.

2. Steps executed by MSS s_j.
   (a) On receiving message *handoff_begin*(h_i) from MSS s_k, transfer MH_DELIV$_i$, MH_SENT$_i$, MH_PENDING$_i$, PEND_ACK$_i$, MH_SSEQNO$_i$ to s_k, and then send message *handoff_over*(h_i) to s_k.

Fig. 4. Handoff module of Algorithm 1.

THEOREM 1. *Algorithm* 1 *ensures causally ordered message delivery.*

PROOF. Note that in Algorithm 1, the MSSs execute the RST algorithm for causal ordering on behalf of MHs. If the mobile hosts do not move, the correctness of the algorithm follows from the correctness of the RST algorithm and from the fact that the wireless channel between an MH and an MSS is FIFO. If an MH h_i moves to another cell, then, during the handoff, all the messages in PEND_ACK$_i$ are first sent to h_i in the FIFO order. Also, h_i retransmits the messages for which it has not received acks from its previous MSS. The data structures MH_DELIV$_i$, MH_SENT$_i$, and MH_PENDING$_i$ are transferred to the new MSS of h_i during the handoff. The new MSS starts executing the causal ordering algorithm only after the handoff terminates (i.e., only after receiving the data structures required to maintain causal ordering). Hence, it is clear that Algorithm 1 ensures causally ordered message delivery. □

4.3 Analysis

For every message sent by MH h_i, the MSS (in whose cell h_i resides) sends MH_SENT$_i$ with the message. Hence, the size

of the header for every message sent over the static network is $O(n_h^2)$ integers. The handoff module uses $O(1)$ messages of size $O(n_h^2)$ numbers when MH h_i switches its cell.

Now, consider the factors F1–F4 discussed in Section 1.1. Since Algorithm 1 is executed at MSSs, factors F1 and F2 are satisfied. The overhead in the wireless medium is kept minimal. But factors F3–F4 are not satisfied. An overhead of $O(n_h^2)$ integers over the static network is costly if n_h is very large. Also, due to disconnections and connections, n_h varies. So, during disconnections, some of the entries in the arrays MH_DELIV, and MH_SENT may be useless. The arrays need not be static, but maintaining dynamic arrays can be complicated if the MH disconnections and connections are frequent. In addition, the processing time for updating the matrix MH_SENT will be substantial for large values of n_h, and the nontrivial processing time increases the delay in delivering a message. These are reflected well in our experimental study shown in Fig. 5. (The details of the simulation model are described at the end of this section.) The average delay experienced by a message is considerably less than the average delay when processing time is taken into account. Algorithm 2, which is presented next, eliminates these disadvantages.

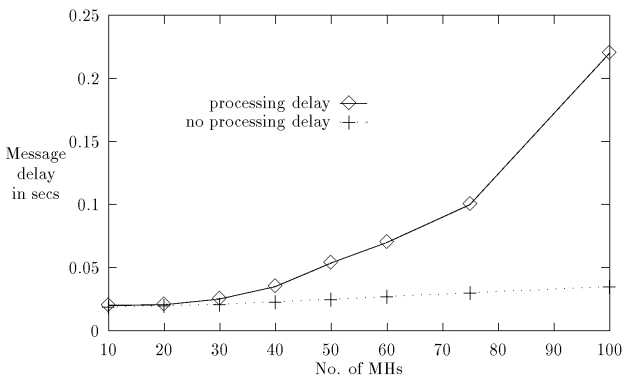


Fig. 5. Average message delay using Algorithm 1 with and without processing delay.

4.3.1 Simulation Details

Our simulation model is similar to that of [11]. The simulation is event driven and it is run on a Sparc 10 station. The events are send message, receive message, and handoff. The bandwidth of a wired channel is assumed to be 100 Mbits/sec, and the propagation delay in a wired channel is 7 ms. For a wireless channel, the bandwidth and propagation delay are assumed to be 1 Mbits/sec and 500 μs, respectively. Initially, the mobile hosts are randomly assigned to the cells. The time interval between two send events in a mobile host is an exponentially distributed random variable with a mean of t_s seconds. The time interval between handoff is also an exponentially distributed random variable with a mean of t_h seconds. The values of t_s and t_h are varied (0.1, 1.0, 10 secs) to consider different scenarios of communication and mobility. The processing time considered in measuring the message delay is the actual CPU running time in processing the message header. The value of every point in the graph is an average of the results of 1,000 experiments performed.

5 Algorithm 2

In Algorithm 1, messages are tagged with complete information to explicitly maintain causal ordering among the mobile hosts. In Algorithm 2, messages are tagged with sufficient information just to maintain causal ordering among the MSSs. Since the wireless channel between an MSS and an MH in its cell is FIFO, maintaining causal ordering at the static network level is sufficient if the MHs do not move between cells. To ensure that causal ordering is not violated after an MH moves between two cells, we incorporate some steps into the handoff procedure.

5.1 Static Module

The static module is similar to the static module of Algorithm 1; the data structures are different. For each MSS s_i, we maintain arrays $\text{MSS\_DELIV}_i[n_s]$, $\text{MSS\_SENT}_i[n_s, n_s]$, and a buffer MSS_PENDING_i. (This is unlike in Algorithm 1 where we maintain these data structures for every mobile host.) Observe that the sizes of the arrays $\text{MSS\_DELIV}_i[n_s]$ and $\text{MSS\_SENT}_i[n_s, n_s]$ vary with n_s, the number of MSSs. The value of $\text{MSS\_DELIV}_i[j]$ indicates the number of messages (whose destinations can be different MHs) received from MSS s_j by MSS s_i. $\text{MSS\_SENT}_i[k, j]$ denotes the number of messages sent by MSS s_k (not necessarily delivered) to MSS s_j that s_i knows of. Every MSS knows (need not be exact) about the location of the MHs. Initially, we assume that the initial locations of MHs are known to all MSSs. In the next section, we show how this knowledge is updated. The static module is formally described in Fig. 6.

Let MHs h_k and h_l be in the cells of MSSs s_i and s_j respectively.

1. On receiving message m from h_k to be sent to the MSS of h_l, MSS s_i executes the following step.
   (a) Send $(m, \text{MSS\_SENT}_i)$ to the MSS s_j.
   (b) $\text{MSS\_SENT}_i[i, j] = \text{MSS\_SENT}_i[i, j] + 1$.
   (c) $\text{MH\_SSEQNO}_k = \text{MH\_SSEQNO}_k + 1$.
   (d) Send an ack to h_k.
2. MSS s_i, on receiving a message for h_k from s_j, executes the following steps.
   (a) If m is not deliverable to h_k, queue (m, ST_m) in MSS_PENDING_i.
   (b) If m is deliverable, then
      i. Transmit m to h_k and queue m in PEND_ACK_k.
      ii. $\text{MSS\_DELIV}_i[j] = \text{MSS\_DELIV}_i[j] + 1$.
      iii. $\text{MSS\_SENT}_i[j, i] = ST_m[j, i] + 1$.
      iv. $\text{MSS\_SENT}_i[k, l] = max(\text{MSS\_SENT}_i[k, l], ST_m[k, l])$, for all k, l
      v. If any message (m, ST_m) in MSS_PENDING_i becomes deliverable, goto step 2(b)i.

Fig. 6. Static module of Algorithm 2.

5.2 Handoff Module

The handoff module is more involved than the handoff module of Algorithm 1. Since causal ordering is explicitly maintained only at the MSS level, some measures are needed during handoff to maintain causal ordering after an MH moves. Before we describe the handoff module, we illustrate the problem at hand with an example.

EXAMPLE. Consider the example shown in Fig. 7. MHs h_1, h_2,

and h_3 are in the cells of MSSs s_1, s_2, and s_3, respectively. Let h_3 send a message m_1 to h_1 (m_1 will be sent to MSS s_1) and then send a message m_2 to h_2. Before receiving m_1, let h_1 switch to the cell of s_2. Now, MH h_2, after receiving m_2 from h_3, sends a message m_3 for h_1 to MSS s_2. If s_2 delivers m_3 to h_1, causal ordering will be violated because h_1 has not yet received m_1. Also, s_2 cannot infer, with the knowledge it has gained so far, whether there are any in-transit messages for h_1 sent to s_1. However, if s_2 delivers m_3 after ensuring that all the messages destined for h_1 sent to s_1 have been delivered to h_1, causal ordering will not be violated.

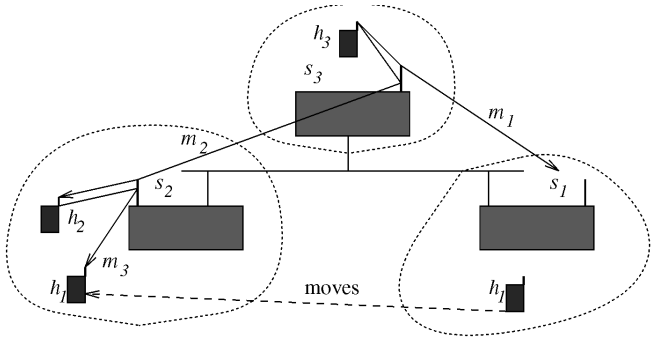


Fig. 7. Example illustrating the violation of causal ordering due to the mobility of hosts.

Next, we describe the handoff module.

Assume that a mobile host h_k switches from the cell of MSS s_i to the cell of MSS s_j. After switching, MH h_k sends *register*(h_k, s_i, MH_RSEQNO$_k$) message to s_j. The message *register*(h_k, s_i, MH_RSEQNO$_k$) to s_j indicates that h_k has switched from the cell of s_i to s_j. On receiving this message, s_j sends the message *handoff_begin*(h_k) to s_i, and then broadcasts the message *notify*(h_k, s_i, s_j) to all the MSSs. The message *notify*(h_k, s_i, s_j) signifies that MH h_k has switched from MSS s_i to MSS s_j. An MSS s, on receiving *notify*(h_k, s_i, s_j) message, updates its local knowledge about the location of MH h_k. MSS s will then send any new messages meant for MH h_k only to s_j (the new MSS of h_k) and not to s_i (the previous MSS of h_k). MSS s also sends a *last*(h_k) message to s_i indicating MSS s_i that messages for h_k will not be sent to MSS s_i anymore.

MSS s_i, on receiving *handoff_begin*(h_k) from s_j sends *enable*(h_k, PEND_ACK$_k$, MH_SSEQNO$_k$) message to s_j and waits for *last*(h_k) messages from all the MSSs. Meanwhile, if any message received by s_i for h_k becomes deliverable to h_k, s_i marks it as old and forwards it to s_j.

Once s_j receives the message *enable*(h_k, PEND_ACK$_k$, MH_SSEQNO$_k$) message, it starts sending the application messages sent by h_k with sequence number greater than MH_SSEQNO$_k$ to their destinations. Also, s_j delivers all the messages in PEND_ACK$_k$ in the FIFO order to MH h_k. Duplication is avoided by delivering only those messages with sequence number greater than MH_RSEQNO$_k$. s_j also delivers all the messages for MH h_k that are marked *old* to h_k in the order in which the messages arrived. If a message for h_k that is not marked *old* (sent by any MSS) becomes deliverable, it is queued and delivered to h_k after the handoff procedure terminates.

MSS s_i (the previous MSS of h_k), after receiving *last*(h_k) from all the MSSs, sends the message *handoff_over*(h_k) to MSS s_j. Observe that no messages for h_k sent to s_i will be in transit after s_i receives *last*(h_k) from all the MSSs. (All messages that are part of handoff procedure are also causally ordered.) The handoff terminates at s_j after *handoff_over*(h_k) is received by s_j. If s_j receives the message *handoff_begin*(h_k) from some other MSS before the current handoff terminates (this can happen if h_k switches it cell), s_j will respond to the message only after the handoff terminates. A description of the handoff module is shown in Fig. 8.

Assume that MH h_k switched from the cell of s_i to s_j.

1. Steps executed by s_j.
   (a) After receiving *register*(h_k, s_i, MH_RSEQNO$_k$) from MH h_k, send *handoff_begin*(h_k) to MSS s_i, and then send *notify*(h_k, s_i, s_j) to all the MSSs.
   (b) On receiving *enable*(h_k, PEND_ACK$_k$, MH_SSEQNO$_k$) message, deliver messages in PEND_ACK$_k$ in FIFO order, and start sending messages sent by MH h_k to their destinations. Use MH_SSEQNO$_k$ and MH_RSEQNO$_k$ to avoid duplication of messages.
   (c) Deliver any messages for h_k from s_i that are marked *old* to h_k. Queue all other messages (not marked old) for h_k that become deliverable and deliver them after the handoff procedure terminates.
   (d) Terminate on receiving *handoff_over*(h_k) from s_i.
2. Steps executed by MSS s_i
   (a) Send *enable*(h_k, PEND_ACK$_k$, MH_SSEQNO$_k$) after receiving *handoff_begin*(h_k) message from s_j. After sending *enable* message drop any message received from h_k.
   (b) If any message for h_k becomes deliverable, mark it as *old* and forward it to s_j
   (c) After receiving *last*(h_k) from all MSSs, send *handoff_over*(h_k) to s_j.
3. Steps executed by all the MSSs.
   (a) On receiving *notify*(h_k, s_i, s_j), update the local knowledge about h_k's location to s_j and send *last*(h_k) to s_i.

Fig. 8. Handoff module of Algorithm 2.

We next prove the correctness of Algorithm 2.

Correctness Argument

Let *mh_send*(m) be the event corresponding to sending of message m by a mobile host. Let *mh_recv*(m) be the event corresponding to the receipt of message m by a mobile host. Let *mss_send*(m) denote the event in an MSS corresponding to the sending of message m (sent by an MH) to another MSS, and *mss_recv*(m) denote the event in an MSS corresponding to the receipt of m sent by another MSS (in which event *mss_send*(m) occurred) to be delivered to an MH.

LEMMA 1. *Let m_1 and m_2 be two messages sent to MH h_k. If* *mh_send*(m_1) $\rightarrow$ *mh_send*(m_2), *then* *mss_send*(m_1) $\rightarrow$ *mss_send*(m_2).

PROOF. First assume that messages m_1 and m_2 are sent by two different MHs. Since *mh_send*(m_1) $\rightarrow$ *mh_send*(m_2), there exists a sequence of events such that *mh_send*(m_1) $\rightarrow$ *mss_send*(m_1) $\ldots \rightarrow$ *mh_send*(m_2). Since *mh_send*(m_2) $\rightarrow$ *mss_send*(m_2), it follows that *mss_send*(m_1)

$\rightarrow$ *mss_send*(m_2). Now consider the case in which m_1 and m_2 are sent by the same mobile host. Assume that the send events *mss_send*(m_1) and *mss_send*(m_2) happened in two different MSSs. (The mobile host may have switched cells in between *mh_send*(m_1) and *mh_send*(m_2).) Otherwise, the result is obvious. Let the mobile host switch from MSS s_1 to MSS s_2. MSS s_2 will send m_2 only after it receives an *enable* message from MSS s_1 as part of handoff. MSS s_1 would have sent m_1 before sending the *enable* message. Hence, *mss_send*(m_1) $\rightarrow$ *mss_send*(m_2). □

THEOREM 2. *Let m_1 and m_2 be two messages sent to MH h_k. If mh_send(m_1) $\rightarrow$ mh_send(m_2), then mh_recv(m_1) $\rightarrow$ mh_recv(m_2).*

PROOF. From Lemma 1, *mss_send*(m_1) $\rightarrow$ *mss_send*(m_2). If m_1 and m_2 are sent to the same MSS, then from the correctness of the RST algorithm it follows that *mss_recv*(m_1) $\rightarrow$ *mss_recv*(m_2). Now, m_1 and m_2 are delivered to MH h_k directly or through handoff (if h_k switches cells). In either case, *mh_recv*(m_1) $\rightarrow$ *mh_recv*(m_2). Now consider the case in which m_1 and m_2 are sent to two different MSSs. Let m_1 be sent to MSS s_i and m_2 be sent to MSS s_j. This can happen only if MH h_k switched from the cell of MSS s_i to the cell of s_j. Now assume that m_1 was not received by MH h_k when it was in the cell of s_i. (Otherwise, h_k would have received m_1 before switching to the cell of s_j, which implies that *mh_recv*(m_1) $\rightarrow$ *mh_recv*(m_2).) Observe that MSS s_j will not deliver m_2 to h_k till the handoff procedure is completed. MSS s_i sends *handoff_over*(h_k) message to s_j only after it receives *last*(h_k) from each MSS. By the time MSS s_i receives *last*(h_k) from all the MSSs, s_i would have forwarded m_1 to s_j or it would have transferred m_1 as part of PEND_ACK$_k$. In either case, s_j will deliver m_1 before the handoff is terminated. Message m_2 will be delivered after the handoff is terminated. Hence, *mh_recv*(m_1) $\rightarrow$ *mh_recv*(m_2). □

THEOREM 3. *A message sent by a mobile host is eventually delivered to its destination.*

PROOF. Let m be a message sent by MH h_k to MH h_l. Let s_i be the MSS of MH h_k and s_j be the MSS of MH h_l. MH h_k first sends message m to its MSS s_i. MSS s_i then send message m to s_j, the MSS of h_l, using RST algorithm. So, from the correctness of RST algorithm, message m eventually gets delivered to MSS s_j. Now, if MH h_l is in the cell of s_j, then s_j will deliver m to MH h_l unless s_j is executing a handoff procedure for h_l. Since a handoff procedure eventually terminates, message m will be delivered to h_l by MSS s_j. If MH h_l has moved to any other MSS, then MSS s_j forwards the message to the new MSS of h_l which will deliver the message immediately to MH h_l. Hence the proof. □

5.3 Analysis

Since the size of MSS_SENT is n_s^2, the size of each message header over the wired network is $O(n_s^2)$ integers. The overhead does not depend on n_h, the number of MHs. Clearly, factors F3-F4 are satisfied. MH connections / disconnections do not affect the size of the arrays MSS_DELIV and MSS_SENT. During handoff, a *notify* message has to be sent to all the MSSs, and all the MSSs send *last* messages. Hence, the handoff module uses $O(n_s)$ messages. The storage requirement of Algorithm 2 and the load placed on the MSSs are less than those of Algorithm 1.

Though the handoff module is involved, it does not affect the performance due to the following reasons.

1) MH h_k does not wait for the handoff module to terminate to receive messages. It continues to receive *old* messages.
2) Messages sent by h_k for other MHs are sent by s_j (the new MSS of h_k) immediately after s_j receives an *enable* message.

The drawback of Algorithm 2 is the possibility of a message being temporarily "inhibited" during delivery to an MH. There is an *inhibition* in delivering a message to an MH if it is queued in MSS_PENDING even though the delivery of the message does not violate causal ordering. Messages may be inhibited because causal ordering is explicitly implemented only among the MSSs. Reception of a message may violate causal ordering from an MSS's point of view, but its delivery to an MH may not violate causal ordering from the MH's point of view. However, this delay is less than the delay introduced by Algorithm 1 in transmitting and processing the large header of each message. The average delay in delivering a message in Algorithm 2 is considerably less than the delay in Algorithm 1 when n_h increases, as shown in Fig. 9. When n_h is less than 30 the message header in both the algorithms are comparable in size. The message delay in Algorithm 2 is more than that of Algorithm 1 due to the inhibition inherent in Algorithm 2. However, as n_h increase the delay due to processing the message header in Algorithm 1 dominates.

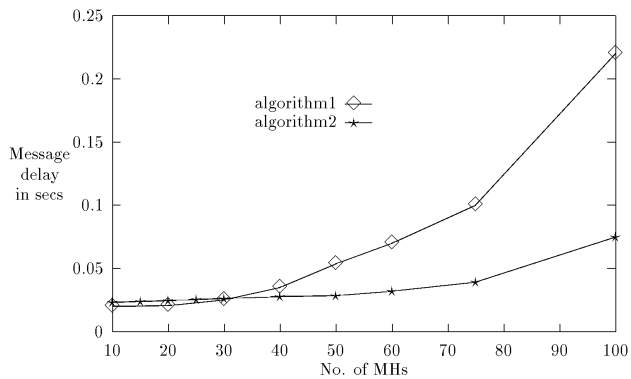


Fig. 9. Comparison of Algorithm 1 and Algorithm 2 with respect to message delay with n_s = 10.

6 ALGORITHM 3

This Algorithm reduces the delay in delivering the messages to MH due to inhibition, the drawback of Algorithm 2, without much increase in the message overhead. The algorithm achieves this by partioning every physical MSS into k logical MSSs.

If an MH enters the cell of an MSS, the MH will be allocated to one of the logical MSS depending on the load in each logical MSS of the MSS. The MHs will communicate with the other MHs through their logical MSSs. Every logical MSS maintains two arrays MSS_DELIV[$k * n_s$] and MSS_SENT[$k * n_s$, $k * n_s$] and a queue MSS_PENDING. The algorithm is the same as Algorithm 2 except for the fact that causal ordering is explicitly maintained among the logical MSSs. The size of the message header is $O(k^2 * n_s^2)$.

Messages to MHs that belong to different logical MSSs will not inhibit each other though they may be in the same cell. Thus, as k increases, the unnecessary delay in delivering the message to MH decreases. However, as k increases, the size of the message header will increase and, as a result, the time to process the message header will become a dominating factor. In Fig. 10, the average message delay initially decreases when k increases. But when k becomes large the average message delay increases.

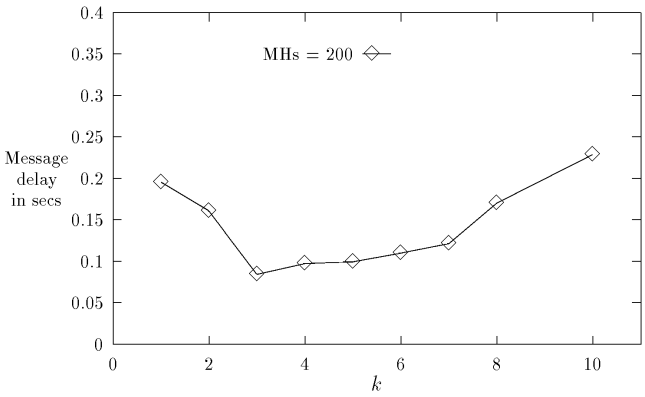


Fig. 10. Message delays for various values of k with $n_s = 10$.

7 Conclusion

7.1 Related Work

There are several algorithms [7], [15], [16] for providing causally ordered message delivery in distributed systems. However, all of these algorithm do not consider host mobility.

Prakash et al. present a causal ordering algorithm for mobile computing environments [14]. They first present an efficient algorithm for static distributed systems. In their algorithm, a message carries information only about its direct predecessor messages with respect to each of its destination process. (Unlike the RST algorithm, information about transitive predecessors is not needed.) By enforcing causal ordering of messages between every pair of immediate causal predecessor and successor messages, causal ordering among all messages is automatically ensured. Thus, by restricting the dependency information carried by a message, the communication overhead can be significantly reduced.

This algorithm is then combined with the algorithm for multicasting in mobile systems [1] to enforce causally ordered multicasting in mobile computing environments. It is interesting to note that Prakash et al.'s algorithm for static distributed systems can be used in the static modules of our algorithms instead of the RST algorithm. This can result in further reduction of communication overhead.

7.2 Concluding Remarks

In this paper, we have considered the problem of causally ordered message delivery to mobile hosts. A direct implementation of an existing algorithm can incur a large message overhead. Algorithm 2 reduces the overhead and it is scalable since the overhead of a message does not vary with the number of mobile hosts. The message overhead can be further reduced by sending the SENT arrays incrementally, using a scheme similar to the one proposed by Singhal and Kshemakalyani for efficient implementation of vector clocks [17]. Algorithm 3 reduces the message delay due to inhibition, inherent in Algorithm 2, by partitioning every MSS into k logical MSSs. The value of k should not be large as a large value of k increases the message overhead.

Our algorithms for causal ordering in mobile systems are based on the RST algorithm for static hosts. Other algorithms can also be modified to work in mobile systems. Only the static modules of our algorithms are based on the RST algorithm. The handoff modules are fairly independent of the RST algorithm. If one wants to handle host mobility in an existing distributed system that supports causal ordering, handoff module of Algorithm 2 can be used with some modification to the existing algorithm that provides causal ordering at the static network level.

Acknowledgments

We would like to thank P. Krishna for providing some basic routines for the simulator. We also like to thank the anonymous referees for their extensive comments which greatly improved the presentation of this paper. A preliminary version of this paper appeared in the Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz, California, December 1994. This research was supported in part by the U.S. National Science Foundation under Grant No. CCR-9110177 and by the Texas Advanced Technology Program under Grant No. 9741-052.

References

[1] A. Acharya and B. Badrinath, "Delivering Multicast Messages in Networks with Mobile Hosts," *Proc. 13th Int'l Conf. Distributed Computing Systems*, pp. 292-299, Pittsburgh, Penn., 1993.

[2] F. Adelstein and M. Singhal, "Real-Time Causal Message Ordering in Multimedia Systems," *Proc. 15th Int'l Conf. Distributed Computing Systems*, pp. 36-43, Vancouver, Canada, 1995.

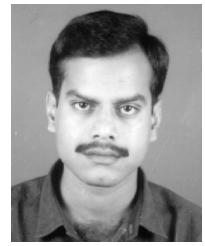
[3] S. Alagar, R. Rajagopalan, and S. Venkatesan, "Tolerating Mobile Support Station Failures," *Proc. Int'l Conf. Fault-Tolerant Systems*, Madras, India, 1995.

[4] A. Athas and D. Duchamp, "Agent-Mediated Message Passing for Constrained Environments," *Proc. First Usenix Symp. Mobile and Location-Independent Computing*, Cambridge, 1993.

[5] B. Badrinath, A. Acharya, and T. Imielinski, "Impact of Mobility on Distributed Computations," *ACM Operating Systems Review*, vol. 27, no. 2, pp. 15-20, 1993.

[6] R. Baldoni, A. Mostefaoui, and M. Raynal, "Efficient Causally Ordered Communication for Multimedia Real-Time Applications," *Proc. Fourth Int'l Conf. High Performance Distributed Computing*, pp. 140-147, Washington, D.C., 1995.

[7] K. Birman and T. Joseph, "Reliable Communications in Presence of Failures," *ACM Trans. Computing Systems*, vol. 5, no. 1, pp. 47-76, 1987.
[8] J. Ioannidis, D. Duchamp, and G. Maguire, "IP-Based Protocols for Mobile Internetworking," *Proc. ACM SIGCOMM Symp. Comm. Architecture and Protocols*, pp. 235-245, Zurich, 1991.
[9] D. Johnson, "Scalable and Robust Internetwork Routing for Mobile Hosts," *Proc. 14th Int'l Conf. Distributed Computing Systems*, Poznan, Poland, 1994.
[10] P. Krishna, N. Vaidya, and D. Pradhan, "Recovery in Distributed Mobile Environments," *Proc. Workshop Advances in Parallel and Distributed Systems*, pp. 83-88, Princeton, N.J., 1993.
[11] P. Krishna, N. Vaidya, and D. Pradhan, "Recovery Issues in Distributed Mobile Environments," Computer Science Technical Report 93-054, Texas A&M Univ., Dec. 1993.
[12] L. Maport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Comm. ACM*, vol. 21, no. 7, pp. 558-565, 1978.
[13] B. Marsh, F. Douglis, and R. Caceres, "System Issues in Mobile Computing," Technical Report MITL-TR-50-93, MITL, 1993.
[14] R. Prakash, M. Raynal, and M. Singhal, "An Efficient Causal Ordering Algorithm for Mobile Computing Environments," *Proc. 16th Int'l Conf. Distributed Computing Systems*, Hong Kong, 1996.
[15] M. Raynal, A. Schiper, and S. Toueg, "Causal Ordering Abstraction and a Simple Way to Implement It," *Information Processing Letters*, vol. 39, no. 6, pp. 343-350, 1991.
[16] A. Schiper, J. Eggli, and A. Sandoz, "A New Algorithm to Implement Causal Ordering," *Proc. Third Int'l Workshop Distributed Algorithms*, pp. 219-232, Berlin, 1989.
[17] M. Singhal and A. Kshemkalyani, "An Efficient Implementation of Vector Clocks," *Information Processing Letters*, vol. 43, pp. 47-52, 1992.
[18] F. Teraoka, Y. Yokote, and M. Tokoro, "A Network Architecture Providing Host Migration Transparency," *Proc. ACM SIGCOMM Symp. Comm. Architecture and Protocols*, Zurich, 1991.
[19] R. van Renesse, "Causal Controversy at Le Mont St.-Michel," *ACM Operating Systems Review*, vol. 27, no. 2, pp. 44-53, 1993.



Sridhar Alagar received a BSc in physics in 1987 from Madurai Kamaraj University and a BE in computer science in 1990 from the Indian Institute of Science. From May 1990 to July 1991, he worked as a software engineer at PSI Data Systems Ltd., Bangalore. In August 1991, he joined the Graduate School, University of Texas at Dallas, and obtained a PhD in computer science in 1995. Since September 1994, he has been a technical staff member at Alcatel Network Systems, where he is working in the area of survivable telecommunication networks. His other research interests include debugging distributed systems, fault-tolerant distributed systems, and mobile computing.

S. Venkatesan received the BTech degree in civil engineering and the MTech degree in computer science form the Indian Institute of Technology, Madras, in 1981 and 1983, respectively, and the PhD degree in computer science from the University of Pittsburgh in 1988. He joined the University of Texas at Dallas in January 1989 where he is currently an associate professor of computer science. His research interests are fault-tolerant distributed systems, survivable telecommunications networks, and mobile computing.