PAPER

A Causal Multicast Protocol for Mobile Distributed Systems

Kuang-Hwei CHI[†], **Li-Hsing YEN**[††], **Chien-Chao TSENG**[†], *and* **Ting-Lu HUANG**[†], ***Nonmembers***

SUMMARY Causal message ordering in the context of group communication ensures that all the message receivers observe consistent ordering of events affecting a group as a whole. This paper presents a scalable causal multicast protocol for mobile distributed computing systems. In our protocol, only a part of the mobility agents in the system is involved in group computations and the resulting size of control information in messages can be kept small. Our protocol can outperform qualitatively the counterparts in terms of communication overhead and handoff complexity. An analytical model is also developed to evaluate our proposal. The performance results show that the proposed protocol is promising.

key words: *causal ordering, multicast, mobile computing*

1. Introduction

One of the fundamental problems in distributed computing is to reduce the nondeterminism due to asynchronous process execution speeds and unpredictable communication delays. To this end, the research community proposed causal ordering for message delivery. That is, for any messages addressed to the same destination, if the sending of a message "happened before" that of another message, then the former message should be received before the latter is. In the context of group communication, the paradigm for such message ordering — causal multicast, ensures that participants observe consistent ordering of events affecting the group as a whole. This simplifies the development of distributed programs for reliable and fault-tolerant systems by providing a built-in message synchronization [11].

The prevalence of portable communication devices and the current trend towards ubiquitous computing have led to the development of a mobile distributed environment. In this setting, hosts are allowed to roam around freely while retaining network connectivity to the system over wireless links, via points of attachment called *mobility agents*. Varying location and tight resource constraints on mobile hosts (MHs), and low bandwidth of wireless connections imply that existing causal ordering algorithms for static hosts are not viable for a mobile environment [1].

The basic idea behind current solutions to causal ordering is that each message is sent with control information about causally preceding messages. A recipient process can thus determine to defer a message from delivery unless all its causally prior messages have been delivered. Such control information, message space overhead, is essentially a form of an integered-vector. In a mobile environment, causal multicast can be achieved by multiple unicasts, using point-to-point protocols in [3], [24]. These protocols demand message space overhead of $O(n_a^2)$ to $O(n_h^2)$ integers, where n_a and n_h, respectively, denote the number of mobility agents and MHs in the system. Prakash et al. proposed an algorithm that allows messages directed to an arbitrary set of mobile hosts [19]. This algorithm is highly flexible, but the incurred control information might still amount to $O(n_h^2)$ integers.

To scale up a system, we can organize mobile hosts and mobility agents into a client-server model, MHs being clients and agents as servers. Ordered message delivery results from serializing messages by hosts within the administrative realm of each agent and using the agents to perform a causal ordering protocol. As a consequence, the amount of control information sent with messages becomes independent of the size of growing multicast groups. Here an issue of concern is how mobility agents are involved in the given protocol so that the vector structure employed need not adapt upon host migrations. In words, the problem is whether all agents, or only those at MHs' current service, should be involved in the protocol execution. In the former case, the vector structure used in the protocol remains unchanged wherever hosts roam, and the hosts' locations need not be tracked. However, each message with $O(n_a)$ space overhead will be sent to all n_a agents, which is unsuitable for Internet-wide deployment. An example of such schemes can be found in [16].

In contrast, if only agents that currently serve MHs are involved, the resulting causal multicast protocol would be more scalable. An additional mechanism is required to maintain the serving agents (called *host view* in [2]) which may change upon host movements. Also, the semantics of group membership (the set of destina-

Manuscript received November 5, 1999.
Manuscript revised July 4, 2000.

[†]The authors are with the Department of Computer Science and Information Engineering, National Chiaotung University, Taiwan.
[††]The author is with the Department of Computer Science and Information Engineering, Chunghua University, Hsinchu, Taiwan.

tions of a multicast message) must be carefully defined when multicast messages and host-view update messages are interlaced, wherein synchronizations need to apply. If MHs' local agents are used to maintain causal ordering, then host view can change frequently as hosts migrate, at the expense of costly synchronizations.

As a remedy, this paper presents a causal multicast protocol that updates host view only when some host joins or leaves a given group, regardless of the participant whereabouts. Hence the number of host-view updates is minimized. For scalability, our protocol follows a client-server model. Multicast messages to and from a mobile host are always relayed through a fixed agent which maintains host view. In our architecture, each message tagged with $O(n_a)$ integers of control information is sent to the agents only specified in host view. A recipient agent then removes the tagged information from each such deliverable message and forwards the resultant part to the MHs in its service list. Our protocol imposes a light load on mobile hosts and wireless links, and could thus by far outperform the existing schemes when only few mobility agents act as servers. Although we may need to redirect messages for a host that is away from its serving agent, this side effect of non-optimal routing is shown to be insignificant to our overall system performance (See Sect. 6.)

The rest of this paper is structured as follows. The system model is introduced in the next section. Section 3 formally presents our protocol and its correctness is proved in Sect. 4. Section 5 summarizes the complexity results of our protocol, in comparison with those of the related work. Further quantitative analyses among the subject protocols are conducted in Sect. 6. Lastly we conclude this paper.

2. Preliminaries

2.1 System Model and Definitions

As illustrated in Fig. 1, we consider a setting that consists of mobile and stationary units interconnected by a static network. The stationary units refer to fixed hosts or mobility agents whose locations do not change over time. A mobility agent, or agent for short, acts as a point of attachment to the network for mobile hosts through wireless links. An MH may be disconnected from the system and reconnected later to reduce power consumptions, or due to movements outside any coverage area of wireless media. Along with migration, an MH alters the attached agent and initiates a *handoff* procedure as follows. When an MH h changes from agent a_i to agent a_j, h registers at a_j with the identity of the previous agent, namely a_i. The new agent a_j then informs a_i of h's switch to a_j. In response, a_i will send the state information associated with h to a_j, upon which the handoff procedure terminates.

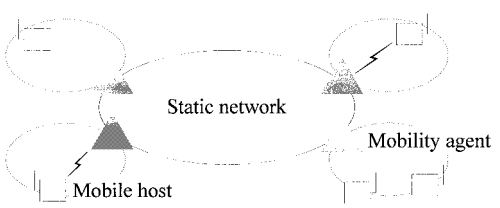


Fig. 1 The model of a mobile distributed system.

The logical channel between any two sites in the system (static or wireless network part) is assumed to be reliable and FIFO, i.e., any message sent by a process is eventually received by non-faulty destination processes. This could result from using the underlying communication primitives introduced in [17].

We assume that a single process runs on a single site. An atomic action that changes the state of a process is defined as an *event*. The execution of a process is modeled by three types of events, namely message send, message receive, and internal events. An internal event, representing a local computation at the process, is unrelated to causal order of messages and is thus not considered. Without mechanisms to perfectly synchronize local clocks in hosts, the temporal ordering of events occurring in hosts can be a partial relation only. One of such partial relations is the *happened-before* relation ($\rightarrow$) defined by Lamport [15] as follows. For any two events e and e', $e \rightarrow e'$ is true if

- e and e' occur in the same process, and e occurred before e',
- e is the sending of message m and e' is the receiving of m, or
- There exists an event e'' such that $e \rightarrow e''$ and $e'' \rightarrow e'$.

In an asynchronous system, messages usually arrive at a process out of causal order. A causal ordering protocol is added to regulate the order of message delivery to the process. For clarity, the term "receive" is distinguished from "deliver": a message is said to be *received* by a process when it arrives at that site, whereas a message is said to be *delivered* by a site when it is formally accepted by the associated application. Let $send(m)$ and $deliv(m)$ denote the sending event and the delivering event of message m, respectively. Causal ordering of message delivery is obeyed if, for any two messages m and m' meant for the same destination, $send(m) \rightarrow send(m')$ implies $deliv(m) \rightarrow deliv(m')$.

2.2 Rated Work

There has been substantial research on causal message ordering. For insightful study and formulation, we refer the reader to [6], [14]. Conventional solutions are centralized or distributed. In centralized schemes [13], [21], a dedicated coordinator serializes all messages exchanged in the system, essentially resulting in a total

message ordering. These scheme can significantly restrict the concurrency in distributed computations, and create a performance bottleneck at the coordinator. On the other hand, in distributed approaches [4], [5], [14], [20], [22], each process can send messages to any others without the intervention of a coordinator. To realize, a common technique that exploits vector clocks [10], [18] or similar variant [9] is deployed: each message is sent with a vector of integers. Such a vector indicates how many messages, and from which senders, causally precede this message. On receipt of the message, a process can thus determine how many messages it must deliver beforehand. Given n_h mobile hosts in the system, a vector of $O(n_h^2)$ integers is shown to be generally necessary for systemwide causal message ordering [14]. This overhead will become intolerable when n_h grows large, limiting a system scalability.

3. The Protocol

We develop a causal multicast protocol in a client-server style. Within the administrative realm of an agent, the agent operates as the local coordinator to serialize multicast messages by MHs. Among agents, a mechanism inspired from the ISIS *cbcast* algorithm by Birman et al. [4], [5] applies. Our protocol is hybrid in the sense that it is a compromise between the centralized scheme and the distributed approach.

3.1 Protocol Overview

We now explain the basic ideas underlying our proposal and how causal message ordering is maintained in our architecture. Consider first a system consisting of one agent and several mobile hosts, as depicted in Fig. 2 (a). Under a single agent a_i, we specify that messages for a group are relayed to a_i which then forwards them in sequence to each intended member. Given $send(m) \rightarrow send(m')$ in the system, a_i will deliver m and then m'. Since the communication channel is FIFO, the property $deliv(m) \rightarrow deliv(m')$ is preserved in each destination host.

Figure 2 (b) demonstrates a case of multiple agents. Assume that messages m and m' are issued by MHs via two distinct agents. Since we specify mobility agents to maintain causal message ordering on behalf of mobile hosts, if $send(m) \rightarrow send(m')$ is present in the system, then this relation is respected in agent perspectives (see Lemma 1 for formal proof.) Furthermore, causal message ordering is enforced at agent level, so each agent will deliver m before delivering m'. Because an agent maintains a FIFO channel with its MHs, the same order of message delivery is honored at each destination host. Note that, in our proposal, an MH always initiates causal multicasts via its serving agent, regardless of its current location. As a consequence, mobile hosts appear stationary, and thus a normal causal ordering algorithm can apply as if it were in a conventional distributed system. This technique of hiding host mobility simplifies the development of our protocol. Hence $deliv(m) \rightarrow deliv(m')$ is preserved everywhere in the system, in the event of host migrations.

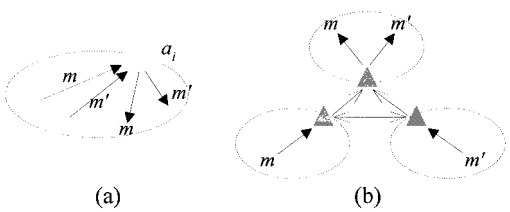


Fig. 2 Illustrating the basic ideas of the proposed protocol.

3.2 Data Structures

The data structures introduced below are all maintained at mobility agents. An MH that applies for a group computation causes its local agent to join the group. Such an agent where a local MH first applies for group computation is called the *serving* agent for the MH. The set of serving agents is denoted as $\mathcal{H}$. The information about $\mathcal{H}$ is distributed only among the members in $\mathcal{H}$ (by some means proposed in Appendix). A serving agent keeps track of each MH h in its service, using a variable loc_h to record the identity of h's currently attached agent. Notice that loc_h is defined only in h's serving agent. It is important to emphasize here that a mobile host's serving agent is not necessarily its local agent.

Each serving agent a_i maintains the following additional data structures:

- a set A_g of serving agents in each multicast group g where a_i participates.
- a vector VT_{a_i} of integers, where $VT_{a_i}[a_j]$ denotes the number of multicast messages known by a_i that agent a_j has initiated.
- a vector VT_h of integers for each MH h served, where $VT_h[a_j]$ counts multicasts by a_j to h.
- an integer SEQ_h per MH h served, indicating the number of multicast messages that h sent and have been received by a_i.

A serving agent will check whether to deliver incoming messages for mobile hosts in the service list. A deliverable message is copied into a FIFO queue $DLVQ_h$ for each applicable MH h; messages in $DLVQ_h$ are transmitted to h in sequence. The transmission is either over a local wireless link or to a remote agent, depending on the value of loc_h.

Our protocol mainly consists of two modules: main module and handoff module. The main module, activated by a message-send operation on a host, enforces causal ordering of multicast message delivery among agents and eventually among the participant MHs. The

handoff module is performed whenever an MH alters its attached agent.

3.3 Main Module

Each mobile host issues multicast messages by always sending them (possibly via the local mobility agent) to its serving agent. Upon receiving a message m from an MH h for group g, the serving agent a_h acknowledges h. MH h can henceforth depart from the sending process. Meanwhile a_h initiates an agent-level causal multicast as follows. First, increment both SEQ_h and $VT_{a_h}[a_h]$ by one, and then set $VT_h[a_h]$ to $VT_{a_h}[a_h]$. Secondly send messages (VT_h, g, m) and VT_h to all the agents in A_g and in $\mathcal{H} - A_g$, respectively. This effectively propagates the delivery information to other groups so that causal message ordering among multigroups is preserved. Thirdly, forward message m to the MHs served by a_h, in the same way as processing a deliverable message, to be described below.

Note that each multicast message sent by an MH h is tagged with VT_h, rather than VT_{a_h}, since VT_h expresses the correct message delivery information with respect to h. Essentially VT_h counts the number of messages, on a per-sending-agent basis, that causally precede m.

An agent a_i, on receiving a message (VT_m, g, m) from some other agent a_j, delays m until

$$VT_m[a_j] = VT_{a_i}[a_j] + 1, \qquad \text{and} \tag{1}$$

$$VT_{a_i}[a_k] \geq VT_m[a_k],\ \forall a_k \in \mathcal{H} - \{a_j\} \tag{2}$$

The first condition ensures that a_i has received all the messages from a_j that precede m, while the second condition ensures that a_i has received all the messages received by a_j before sending m. If the above two conditions hold, m is deliverable at a_i (because there has been no causally prior message that should be delivered.) Agent a_i subsequently updates $VT_{a_i}[a_j]$ to $VT_m[a_j]$ and proceeds the MH-level delivery: for each MH h served by a_i, enqueue m in $DLVQ_h$. A thread may be allocated to transmit messages in $DLVQ_h$ to h in FIFO order. After MH h acknowledges the receipt of m, a_i updates $VT_h[a_j]$ to $VT_m[a_j]$.

As noted, a timestamp associated with each multicast is sent to the agents in $\mathcal{H} - A_g$. A recipient agent handles such a control message as if it were a normal message to be delivered, except that the MH-level delivery is skipped. The control message is used to notify these agents of the existence of the multicast message, which is essential for our protocol to operate in a multigroup environment.

A formal description of the main module is given in Fig. 3.

3.4 Handoff Module

A handoff procedure takes place whenever an MH discovers its movement to a new local agent. Subsequent interactions follow among the previous agent, the serving agent, and the new agent of the MH, as described in Fig. 4. (The serving agent identity will be available by contacting the previous agent.) An important task to be performed is that the serving agent updates its knowledge about the MH's new location.

Let a_h be the serving agent of MH h.

1. On receiving a message m from h for group g, agent a_h executes the steps below.
   a. $SEQ_h := SEQ_h + 1$.
   b. Set both $VT_h[a_h]$ and $VT_{a_h}[a_h]$ to $VT_{a_h}[a_h] + 1$.
   c. Send (VT_h, g, m) and VT_h to all the agents in A_g and $\mathcal{H} - A_g$, respectively.
   d. Perform Step 2c. /* m is deliverable at a_h, so deliver m to the MHs served by a_h */
2. On receipt of a message m, timestamped with VT_m, from agent a_j for group g, agent $a_i \neq a_j$ performs the following.
   a. Delay message m until it is deliverable at a_i.
   b. $VT_{a_i}[a_j] := VT_m[a_j]$.
   c. For each MH h served by a_i, do the following.
      i. Insert m to $DLVQ_h$.
      ii. After receiving an ack for m from h, update $VT_h[a_j]$ to $VT_m[a_j]$.
   d. If any delayed message becomes deliverable, goto Step 2b.
3. On receiving a message m with timestamp only, a_i performs Steps 2a and 2b. /* the message contains null data to be delivered */

Fig. 3 Main module.

Let an MH h hand off from a_i to a_j.

1. Steps executed by agent a_j.
   a. On receiving $register(h, a_i)$ from h, if loc_h is undefined, then relay the message to a_i; otherwise perform Steps 3(a)i and 3(a)ii.
   b. On receiving $reg\_ack(h, SEQ_h, a_j)$ from a_h, conduct the following actions.
      i. Send $reg\_ack(SEQ_h)$ to h.
      ii. Record a_h as h's serving agent.
2. Steps executed by a_i. /* the previous agent of h */
   a. On receiving $register(h, a_i)$ from a_j, if loc_h is undefined, then relay this message to h's serving agent; otherwise perform Steps 3(a)i and 3(a)ii.
   b. On receiving $reg\_ack(h, a_j)$, delete the relevant data structures for MH h, if loc_h is undefined.
3. Steps executed by a_h. /* h's serving agent */
   a. On receiving $register(h, a_i)$ from a_j, do the following.
      i. Set loc_h to a_j. /* a_i was stored previously */
      ii. Send $reg\_ack(h, SEQ_h, a_j)$ to both a_i and a_j.

Fig. 4 Handoff module.

Let an MH h move and alter its attached agent from a_i to a_j. To begin with the handoff procedure, MH h issues a registration message $register(h, a_i)$ to the new local agent. If the variable loc_h is undefined in a_j, then a_j is not h's serving agent, so the registration message is transacted with a_i. The previous agent a_i of h, after receiving the registration message, also checks whether a_i itself is h's serving agent, and will pass this message to the intended serving agent (if necessary.) Suppose that a_h is the serving agent of h. Agent a_h updates loc_h to a_j, and then sends message $reg\_ack(h, SEQ_h, a_j)$ to h's previous agent and new agent parallely. This acknowledgment causes a_i to revoke the obsolete information for h properly, and a_j to inform h of SEQ_h, upon which the handoff procedure terminates. From SEQ_h, MH h can decide which message should be retransmitted next.

It can be seen that along with an MH migration, its serving agent identity will be propagated to the MH's currently attached agent.

3.5 Handling Disconnections and Reconnections

To reduce the battery power consumption, an MH may disconnect itself from the system volunteerly. To begin, an MH, say h, sends a $disconnect(h)$ message to its local agent which then informs h's serving agent, a_h, of the disconnection. Within a_h, all the multicast messages meant for h are processed as if h were still active in the group, except that message delivery to the MH is suppressed. When h is reconnected to the system later by sending a $reconnect(h)$ message to a_h, messages in $DLVQ_h$ is delivered to h. If h changes the attached agent prior to the reconnection, a handoff procedure must have been performed before h is able to receive or send any multicast messages.

To avoid an indefinite disconnection of an MH, a serving agent runs a timer for each MH served. This can be used to detect the MHs that were not reconnected to the system in time, or that have not responded to the MH-level message delivery for a long time. When the timer expires, the MH is treated as having departed from the system and the serving agent informs the group administrator of the MH's departure. (See Appendix.) Then flushing follows, in the same way as the MH requests to leave the interested group(s).

3.6 Remarks

The vector timestamped on each message in our protocol is $|\mathcal{H}|$ integers; only $|\mathcal{H}|$ of the n_a mobility agents in the system are used to maintain causal message ordering for MHs. We conclude that, taking into account the number of destination agents of a message, totally our protocol uses $|\mathcal{H}|^2$ integers to convey control information during a multicast.

In our protocol, the delivery information for different groups is aggregated within VT_{a_i} in each serving agent a_i. This may lead to a situation where two messages are initiated concurrently for two distinct groups, but delayed at a recipient agent because one of the messages is misinterpreted to be causally preceding the other one. The concurrency within distributed applications can be thus restricted. Fortunately, Alagar and Venkatesan showed that such delay is not serious [3]; the disadvantage of unnecessary delay will be cancelled out by the benefits gained from the reduction in message space overhead. Indeed, there is a tradeoff between message overhead and unnecessary delay.

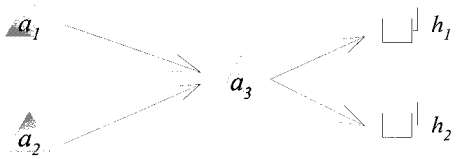


Fig. 5 An example of duplicated message delivery.

We use a message redirection technique to fix the number of participant agents in group communication. A message redirection takes place when a sending MH or a destination MH is not attached to its serving agent locally, resulting in a longer communication latency. However this does not impose a significant flaw to our scheme, since, as shown in considerable research like [8], MHs have locality in their movements, that is, each MH is likely to remain attached to its serving agent.

In our protocol, duplicated multicast messages can be sent to some mobility agents, as illustrated in Fig. 5. Suppose that MHs h_1 and h_2 in a group are served by agents a_1 and a_2, respectively. When there is a multicast message meant for the group, the agent a_3 that are attached by both MHs will receive two identical copies of the message, each delivered from a_1 and a_2 separately. The duplication of messages does not impair the correctness of our protocol and can be avoided by using a scheme presented in [7].

4. Correctness Proof

The correctness of the proposed protocol is proved in stages. We will show that causality is never violated (safety property) and then demonstrate that our protocol never defers a message indefinitely (liveness property). The reasoning about the proof here is mainly adapted from [5].

Let $mh\_send(m)$ be the sending event of message m by a mobile host. And, $ma\_send(m)$ denotes the event in an agent corresponding to the sending of message m (sent by the MH) to another agent.

Lemma 1: Let m and m' be two messages directed to a mobile host. If $mh\_send(m) \rightarrow mh\_send(m')$, then $ma\_send(m) \rightarrow ma\_send(m')$.

Proof: First consider the case in which m and m' are

sent by the same MH. The result is obvious because the logical channel between the MH and its serving agent is FIFO (regardless of the MH's movements); message m will have been sent by the serving agent before m' is. Thus $ma\_send(m) \rightarrow ma\_send(m')$.

Now assume that messages m and m' are sent by two different mobile hosts. Since $mh\_send(m) \rightarrow mh\_send(m')$, there exists a sequence of events such that $mh\_send(m) \rightarrow ma\_send(m) \rightarrow \cdots \rightarrow mh\_send(m')$. Because $mh\_send(m') \rightarrow ma\_send(m')$, it follows that $ma\_send(m) \rightarrow ma\_send(m')$. □

Theorem 1 (safety property): Suppose messages m and m' such that $mh\_send(m) \rightarrow mh\_send(m')$ are destined for an MH h. As far as h is concerned, m' cannot be delivered unless m has been delivered.

Proof: From Lemma 1, $ma\_send(m) \rightarrow ma\_send(m')$. Consider the actions of an agent a_i that receives the two messages. There are two possible cases to be addressed.

If m and m' are both initiated by the same agent, say a_j, then $VT_m[a_j] < VT_{m'}[a_j]$. From Step 2b of our protocol, if m has not been delivered at a_i, it follows that $VT_{a_i}[a_j] < VT_m[a_j]$. Hence $VT_{a_i}[a_j] \neq VT_{m'}[a_j] - 1$ and m' is deferred from delivery by condition (1) of our protocol.

Now consider the case in which m and m' are initiated by two different agents. We next show that m' cannot be delivered before m by induction on the messages delivered at a_i. From the definition of the *happened-before* relation, it can be shown that $send(m) \rightarrow send(m')$ implies $VT_m < VT_{m'}$. In particular, considering the initiator a_j of m, we have the relation

$$VT_m[a_j] \leq VT_{m'}[a_j]. \tag{3}$$

Base step. The first message delivered at a_i cannot be m'. If no messages have been delivered at a_i, then $VT_{a_i}[a_k] = 0, \forall a_k$. However, $VT_m[a_j] > 0$ (because m is initiated by a_j), hence $VT_{m'}[a_j] > 0$. Message m' cannot be delivered at a_i due to condition (2).

Inductive step. Suppose that n messages have been delivered at a_i and none of them is a message m'' such that $send(m) \rightarrow send(m'')$. If m has not yet been delivered, then from Lemma 1, we have the relation

$$VT_{a_i}[a_j] < VT_m[a_j]. \tag{4}$$

From relations 3 and 4, it follows that $VT_{a_i}[a_j] < VT_{m'}[a_j]$. Because m' does not satisfy condition (2), it cannot be the $(n+1)$st message to be delivered at a_i.

The above two cases show that m must be delivered at a_i before m' is. Since the channel between h and its serving agent is FIFO, m' cannot be delivered to h unless m has been delivered. In other words, causal message ordering among MHs is preserved. □

Theorem 2 (liveness property): A message sent by a mobile host is eventually delivered to its destination.

Proof: Suppose there exists a multicast message m sent by an MH that can never be delivered. Let a_h be the serving agent of the MH and a_i be a recipient agent that defers m indefinitely. For such an agent a_i, the deferring arises from

$$VT_m[a_h] \neq VT_{a_i}[a_h] + 1, \text{ or}$$
$$\exists a_k \neq a_h, VT_m[a_k] > VT_{a_i}[a_k].$$

In the first case, $VT_m[a_h] \neq VT_{a_i}[a_h] + 1$ indicates that m is not the next message to be delivered from a_h to a_i. Since all messages are sent to all the participant agents and the communication channels are reliable, thus there must be a message m' initiated by a_h that a_i has received but not yet delivered and m' is the next message from a_h to be delivered. In other words, $VT_{m'}[a_h] = VT_{a_i}[a_h] + 1$. If m' is also delayed, it should be under the case below.

Next consider the case where $\exists a_k \neq a_h, VT_m[a_k] > VT_{a_i}[a_k]$. If $VT_m[k] = x$, the xth message m' initiated by a_k should have been sent before m was sent. Message m' either has not yet been received or has been received but deferred by a_i. Since the communication channels provide lossless message transmission, m' is received eventually by a_i. The reasoning that was applied to m is also applicable to m'. This leads to a contradiction, because the number of messages that must be delivered before m is finite.

The above shows that a message is never deferred at any agent indefinitely. Each deliverable message is transmitted to the destination MHs from their respective serving agents, although some of the MHs may be executing handoff procedures. The handoff procedures eventually terminate and these MHs will be delivered messages subsequently. Hence the proof. □

5. Complexity Results

Table 1 shows the complexity results of the existing causal ordering protocols for mobile hosts, where k is the number of logical agents per agent [3]. For abbreviation we denote the Li-Huang algorithm [16], the Prakash-Raynal-Singhal algorithm [19], and the Yen-Huang-Hwang algorithm [24] by LH, PRS, and YHH, respectively. In [3], Alagar and Venkatesan proposed three algorithms which in sequence are denoted by AV-1, AV-2 and AV-3, respectively. Note that YHH and the algorithms by Alagar and Venkatesan are mainly designed for point-to-point communication paradigm.

Communication overhead denotes the number of integers of a vector timestamp when sending a message in a causal ordering protocol. This dimension, reflecting the additional bandwidth used by the control information per message, is determined by two factors: the vector size on each message and the number of destination sites of such a timestamped message. These two factors are used to label the second and the third

Table 1 Complexity results. (Assume $n_h \gg n_a$ [2], where n_h and n_a, respectively, denote the number of MHs and mobility agents in the system.)

Protocol	Communication overhead		Storage overhead	Handoff complexity	
	Message space overhead	No. of destination sites		No. of messages	Message size
AV-1	$O({n_h}^2)$	$O(1)$	$O({n_h}^2)$	$O(1)$	$O({n_h}^2)$
AV-2	$O({n_a}^2)$	$O(1)$	$O(1)$	$O(n_a)$	$O(1)$
AV-3	$O({n_a}^2 \times k^2)$	$O(1)$	$O({n_a}^2 \times k^2)$	$O(n_a \times k)$	$O(1)$
YHH	$O(n_a \times n_h)$	$O(1)$	$O(n_a \times n_h)$	$O(1)$	$O(n_a \times n_h)$
LH	$O(n_a)$	$\Theta(n_a)$	$\Theta(n_a)$	$O(1)$	$\Theta(n_a)$
PRS	$O({n_h}^2)$	$O(n_h)$	$O({n_h}^2)$	$O(1)$	$O({n_h}^2)$
Ours	$O(n_a)$	$O(n_a)$	$O(n_a)$	$O(1)$	$O(1)$

columns, respectively, in Table 1. As a note, in point-to-point protocols each message is intended for a single site, whereas multicast protocols LH, PRS, and ours allow a message to be destined for multiple sites.

To be precise, in our protocol, the message space overhead is $|\mathcal{H}|$ and each timestamped message is meant for $|\mathcal{H}|$ agents. Here a formal statement can be made with $|\mathcal{H}|$ in place of $O(min(n_h, n_a))$. From Table 1, it can be seen that our protocol outperforms the other counterparts in terms of communication overhead.

Storage overhead refers to the size of the data structures that an agent maintains for MHs to execute a given protocol. As described in Sect. 3.2, in our scheme, a serving agent maintains delivery information for MHs, at the expense of $O(min(n_h, n_a))$ integers.

Handoff complexity indicates the amount of information exchanged between agents during a handoff procedure. In our protocol, two messages containing several scalars is transferred between two agents, so $O(1)$ messages of size $O(1)$ are needed during the handoff procedure. This concludes that our protocol also outperforms the other counterparts regarding handoff complexity.

6. Performance Analysis

This section evaluates causal ordering protocols in terms of the average communication latency over the static network. This metrics mainly consists of message propagation time and the delayed time of the message (which is deferred from delivery at the recipient agent.)

6.1 Assumptions

The assumptions used in this performance study are listed below.

A1. Message propagation time between any two mobility agents is exponentially distributed with the same probability density function $f(x) = \beta e^{-\beta x}$.

A2. Message propagation time is approximately proportional to the amount of the control information conveyed. Hence protocols with different message overheads will yield distinct β's.

A3. Compared with message propagation time, both message transmission latency and protocol processing delay are insignificant.

A4. As a message is delivered at an agent, the agent initiates a new causal multicast without delay.

Assumptions A1, A2, and A3 suggest a relation between the averaged message propagation time and the vector timestamp size of messages. That is, $\frac{1}{\beta} \approx$ (message space overhead) $\times$ (message propagation delay per integer).

6.2 Analytical Model

Consider a sequence σ of messages $m_{(k)}, m_{(k-1)}, \cdots, m_{(0)}$ satisfying: (1) $send(m_{(k)}) \rightarrow send(m_{(k-1)}) \rightarrow \cdots \rightarrow send(m_{(0)})$, and (2) for any two consecutive messages $m_{(i)}$ and $m_{(i-1)}$ in σ, there exists no other message m' such that $m_{(i)} \rightarrow m'$ and $m' \rightarrow m_{(i-1)}$. In essence, σ represents a set of sending events over which the *happened-before* relation is defined with the smallest transitivity. Here $m_{(k)}$ is said to be the kth predecessor for $m_{(0)}$. Further, we say that σ is terminated with length $k+1$, denoted by $|\sigma|$, if $m_{(k)}$ has no undelivered predecessor.

Let $X_{(i)}$ be the propagation time for message $m_{(i)}$. The sum $Y_{k+1} = X_{(0)} + X_{(1)} + \cdots + X_{(k)}$ is known to be Erlang distributed with probability density function $g(y) = \frac{\beta(\beta y)^k}{k!} e^{-\beta y}$. The probability that a message, say $m_{(0)}$, waits for the arrival of its kth predecessor is thus

$$
\begin{aligned}
&P[X_{(k)} > X_{(0)} + X_{(1)} + \cdots + X_{(k)}] \\
&\quad = P[X_{(k)} > Y_{k+1}] \\
&\quad = \int_{y=0}^{\infty} \int_{x=y}^{\infty} f(x) g(y) dx dy \\
&\quad = \int_{y=0}^{\infty} [-e^{-\beta x}] \Big|_{x=y}^{\infty} \frac{\beta(\beta y)^k}{k!} e^{-\beta y} dy \\
&\quad = \int_{y=0}^{\infty} e^{-\beta y} \frac{\beta(\beta y)^k}{k!} e^{-\beta y} dy \\
&\quad = \frac{1}{2^{k+1}}. \qquad (5)
\end{aligned}
$$

Notice that Eq. (5) above also indicates the probability

that $|\sigma|$ exceeds k. Therefore we have

$$P[|\sigma| = k+1] = \frac{1}{2^k} - \frac{1}{2^{k+1}} = \frac{1}{2^{k+1}}. \quad (6)$$

At a destination site, message $m_{(i-1)}$ may arrive prior to its immediate predecessor $m_{(i)}$ by $W_{i,i-1}$ time units. From assumption A1, the message propagation time being *memoryless*, it can be shown that, for $w \geq 0$, $P[X_{(k)} \geq Y_{k+1} + w | X_{(k)} > Y_{k+1}] = P[X_{(k)} \geq w] = e^{-\beta w}$. Namely $P[W_{i,i-1} \geq w | W_{i,i-1} \geq 0] = e^{-\beta w}$. When $k = 1$, this equation can be used to derive the expected waiting time of a message for its immediate predecessor, given that the predecessor is yet absent, as follows.

$$\int_0^\infty P[X_{(1)} - Y_2 \geq w | X_{(1)} > Y_2] dw = \frac{1}{\beta}$$
$$= E[W_{i,i-1} | W_{i,i-1} > 0]. \quad (7)$$

If $|\sigma| = k+1$, the delayed time for message $m_{(0)}$ can be expressed as the sum $W_{k,k-1} + W_{k-1,k-2} + \cdots + W_{1,0}$, where $W_{i,i-1} > 0$ $(\forall i > 0)$. Since $W_{i,i-1}$'s are identical independently distributed, for a message we have the conditional expectation

$$E[\text{delayed time} \Big| |\sigma| = k+1]$$
$$= \sum_{i=1}^{k} E[W_{i,i-1} | W_{i,i-1} > 0]$$
$$= \frac{k}{\beta}. \quad (8)$$

In combination with Eq.(6), it follows that

$$E[\text{delayed time}]$$
$$= \sum_{k=1}^{\infty} E[\text{deferred time} \Big| |\sigma| = k+1] P[|\sigma| = k+1]$$
$$= \sum_{k=1}^{\infty} \frac{k}{\beta} \cdot \frac{1}{2^{k+1}}$$
$$= \frac{3}{\beta}.$$

To summarize, the average communication latency of previous protocols can be expressed as a form of $\frac{1}{\beta} + \frac{3}{\beta}$. In contrast, our scheme would demand a latency of $p(\frac{1}{\beta}) + \frac{1}{\beta} + \frac{3}{\beta} + p(\frac{1}{\beta})$, where p denotes the probability that an MH is away from its serving agent. The first term and the last term in the foregoing expression, respectively, represent the extra time to redirect a message to and from an MH at some remote site.

6.3 Performance Results

The network parameters used in our evaluation are instantiated as follows. n_a is 100, and p is 0.3 [8]. Assume that, on average, our message space overhead is $(1-p) \times min(n_a, n_h)$ integers (this assumption could be justified when hosts move randomly.) In addition, the average message space overhead in the PRS algorithm is assumed to be $0.05n_h^2$ integers.

We can derive the ratio of communication latency of our scheme to that of the PRS protocol as $(2p+4)((1-p) \times min(n_a, n_h))/(4 \times 0.05n_h^2)$. Likewise the ratio of our protocol to that of the LH algorithm is $(2p+4)((1-p) \times min(n_a, n_h))/(4n_a)$. Given that n_h is 20, these two ratios result in 80% and 16.1%, respectively. This corresponds to a case where only small groups of mobile hosts are present in the system. When n_h is 120, these two ratios are 11.2% and 80.5%, respectively. The performance of our protocol relative to the counterparts under different parameter settings can also be obtained similarly. The performance results show that our scheme is promising.

7. Conclusion

This paper presented a scalable protocol for causally ordered message delivery among a group of mobile hosts. Our protocol appends to messages the delivery information about the serving agents only. The resulting communication overhead can be thus kept small. An inevitable drawback to our scheme is unnecessary deferring of messages, as discussed in Sect. 3.6. In other words, the proposed protocol achieves better scalability, while at the expense of longer delay in delivering messages. However, some researchers showed that this drawback will be cancelled out by the benefits gained from the reduction in message space overhead.

In addition, our protocol deals with mobile host disconnections and reconnections, and tolerates mobile host stop-failures. A stop-failure could be used to model an MH moving outside any coverage area of wireless media. Under this circumstance, the MH is treated as having been terminated. The correctness of the proposed protocol was proved in this paper. Our protocol outperforms the counterparts in terms of communication overhead and handoff complexity. In Sect. 6, we developed an analytical model to evaluate causal ordering protocols. The performance results demonstrated that our proposal is promising.

Acknowledgments

We thank the anonymous referees for their valuable comments. This work has been supported by the National Science Council, ROC, under grants NSC88-2213-E009-081 and NSC89-2213-E-009-028, and by the MOE Program of E. Research under grant 89-E-FA04-1-4.

References

[1] A. Acharya, "Structuring distributed algorithms and services for networks with mobile hosts," Ph.D. Thesis, Rut-

gers, The State University of New Jersey, May 1995.
[2] A. Acharya and B.R. Badrinath, "A framework for delivering multicast messages in networks with mobile hosts," ACM/Baltzer Mobile Networks and Applications, vol.1, no.2, pp.199–219, 1996.
[3] S. Alagar and S. Venkatesan, "Causal ordering in mobile distributed systems," IEEE Trans. Comput., vol.46, no.4, pp.353–361, 1997.
[4] K.P. Birman and R. van Renesse, Reliable Distributed Computing with the Isis Toolkit, IEEE Computer Society Press, 1994.
[5] K.P. Birman, Building Secure and Reliable Network Applications, Manning Publications Co., 1996.
[6] B. Charron-Bost, G. Tel, and F. Mattern, "Synchronous, asynchronous, and causally ordered communication," Distributed Comp., vol.9, no.4, pp.173–191, 1996.
[7] V. Chikarmane, C.L. Williamson, R.B. Bunt, and W.L. Mackrell, "Multicast support for mobile hosts using mobile IP: Design issues and proposed architecture," ACM/Baltzer Mobile Networking and Applications, vol.3, no.4, pp.365–379, Jan. 1999.
[8] G. Cho and L.F. Marshall, "An efficient location and routing scheme for mobile computing environment," IEEE J. Selected Areas in Commun., vol.13, no.5, pp.868–879, 1995.
[9] D. Dolev and D. Malki, "The transis approach to high availability cluster communication," Commun. ACM, vol.39, no.4, pp.64–70, April 1996.
[10] J. Fidge, "Timestamps in message-passing systems that preserve the partial ordering," Proc. 11th Australian Comput. Science Conf., pp.56–66, Feb. 1988.
[11] V. Hadzilacos and S. Toueg, "Fault-tolerant broadcasts and related problems," Distributed Systems, 2nd ed., ed. S. Mullender, pp.97–145, ACM Press, 1993.
[12] I.H. Jang, J.W. Cho, and H. Yoon, "An efficient causal multicast algorithm for distributed system," IEICE Trans. Inf. & Syst., vol.E81-D, no.1, pp.27–36, Jan. 1998.
[13] M.F. Kaashoek, A.S. Tanenbaum, S.F. Hummel, and H.E. Bal, "An efficient reliable broadcast protocol," Oper. Syst. Rev., vol.23, no.4, pp.5–19, 1989.
[14] A.D. Kshemkalyani and M. Singhal, "Necessary and sufficient conditions on information for causal message ordering and their optimal implementation," Distributed Computing, vol.11, no.2, pp.91–111, 1998.
[15] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," Commun. ACM, vol.21, no.7, pp.538–565, 1978.
[16] C.P. Li and T.L. Huang, "A mobile-support-station-based causal multicast algorithm in mobile computing environment," Proc. 11th Int'l Conf. Info. Networking, vol.2, pp.9C-1.1-9C-1.10, Taipei, Taiwan, Jan. 1997.
[17] K. Obraczka, "Multicast transport protocols: A survey and taxonomy," IEEE Commun. Mag., vol.36, no.1, pp.94–102, 1998.
[18] F. Mattern, "Virtual time and global states of distributed systems," Proc. Int'l Wksp. Parallel and Distributed Algorithms, pp.215–226, North-Holland, 1989.
[19] R. Prakash, M. Raynal, and M. Singhal, "An adaptive causal ordering algorithm suited to mobile computing environments," J. Parallel and Distributed Computing, vol.41, no.2, pp.190–204, March 1997.
[20] M. Raynal, A. Schiper, and S. Toueg, "Causal ordering abstraction and a simple way to implement it," Inform. Process. Lett., vol.39, no.6, pp.343–350, 1991.
[21] M. Reiter and L. Gong, "Securing causal and relationships in distributed systems," The Comput. Journal, vol.38, no.8, pp.633–642, 1996.
[22] A. Schiper, J. Eggli, and A. Sandoz, "A new algorithm to implement causal ordering," Proc. 3rd Int'l Wksp. Distributed Algorithms, 1989. Also published in Lecture Notes in Computer Science, p.392.
[23] L.H. Yen and T.L. Huang, "Resetting vector clocks in distributed systems," J. Parallel and Distributed Computing, vol.43, pp.15–20, 1997.
[24] L.H. Yen, T.L. Huang, and S.Y. Hwang, "A protocol for causally ordered message delivery in mobile computing systems," ACM/Baltzer Mobile Networks and Applications, vol.2, no.4, pp.365–372, 1997.

Appendix: Group Management

Adding or dropping a host from a group in the middle of message exchanges among the group members requires a form of synchronizations [4, pp.109–132]. The messages that were sent by each participant in an original group, before the group changes, need to be flushed to their destinations. Hence we say that a *flushing* is initiated within x, given a group x of sites. Flushing can be dealt with by efficient solutions, e.g., the scheme by Yen and Huang [23]. Such a proposal is used as a building block to manage groups in our protocol.

Assume that there is an administrator site which maintains the information about $\mathcal{H}$ and A_g per multicast group, and authenticates the hosts that apply for participation in a group. The administrator is known beforehand to the interested hosts. To join group g, MH h sends a join-group request via the current agent, say a_h, to the administrator for permission. If h is admitted, the administrator then conducts the actions shown in Fig. A· 1. In response to the acknowledgment from the administrator, a_h will execute some necessary steps such as depositing of the information of $\mathcal{H}$ and A_g, initializing VT_h to VT_{a_h}, allocating storage space for $DLVQ_h$, and then acknowledging h. At this point,

Let a_h be the local agent of an MH h that requests to join multicast group g.

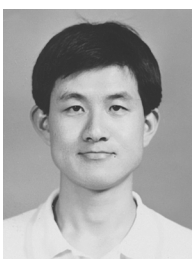
1. If $a_h \in \mathcal{H}$, then do the following.
   a. If $a_h \notin A_g$, then
      i. Initiate a flushing within A_g.
      ii. Inform all the agents in A_g of a directive saying "add a_h to A_g".
   b. Acknowledge a_h for h is admitted.
2. If $a_h \notin \mathcal{H}$, then perform the steps below.
   a. Initiate a flushing within $\mathcal{H}$.
   b. Inform all the agents in A_g of a directive saying "add a_h to both $\mathcal{H}$ and A_g", and all the agents in $\mathcal{H} - A_g$ of a directive saying "add a_h to $\mathcal{H}$".
   c. Send the information about $\mathcal{H}$ and A_g to a_h as an acknowledgment.
   d. Record the binding between a_h and h.

Fig. A· 1 Actions executed by the administrator upon receiving a join-group request.

a_h start as a serving agent for h.

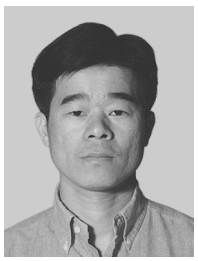
Notice that after a flushing within $\mathcal{H}$, each agent in $\mathcal{H}$ will receive a directive from the group administrator, upon which its delivery vectors are caused to add an entry for a_h and reset to all zeros.

On the contrary, when MH h requests to leave group g, the request is also relayed to the administrator. The recorded bindings are examined to determine whether there exists any other MH whose serving agent is a_h. If so, but none of the remaining MHs belongs to A_g, the administrator initiates a flushing with an additional directive saying "remove a_h from A_g." If not, a flushing is initiated with an additional directive saying "remove a_h from both $\mathcal{H}$ and A_g." The directive causes each of the recipient agent to adjust its delivery vectors and the knowledge about A_g (or $\mathcal{H}$) accordingly. In all cases, a_h will revoke the data structures for h from the local storage.



Kuang-Hwei Chi received his B.S. degree in Computer Science and Information Engineering from Tatung Institute of Technology, Taipei, Taiwan, in 1991 and his M.S. degree in Computer Science and Information Engineering from National Chiao-Tung University, Hsinchu, Taiwan, in 1993. He is currently a Ph.D. candidate in the Department of Computer Science and Information Engineering from National Chiao-Tung University. He is a member of the ACM.

Li-Hsing Yen is currently an assistant professor at the Department of Computer Science and Information Engineering in Chung-Hua University, Hsin-Chu, Taiwan. He received his BS, MS, and Ph.D. degrees in Computer Science and Information Engineering, all from National Chiao-Tung University, Hsin-Chu, Taiwan, in 1989, 1991, and 1997, respectively. His research interests include distributed algorithms, wireless communications, and mobile computing.



Chien-Chao Tseng is currently a professor in the Department of Computer Science and Information Engineering at National Chiao-Tung University, Hsin-Chu, Taiwan. He received his BS degree in Industrial Engineering from National Tsing-Hua University, Hsin-Chu, Taiwan, in 1981; MS and Ph.D. degrees in Computer Science from the Southern Methodist University, Dallas, Texas, USA, in 1986 and 1989, respectively. His research interests include mobile computing, and wireless Internet.

Ting-Lu Huang studied at Tunghi 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.