Proc. Natl. Sci. Counc. ROC(A)
Vol. 23, No. 1, 1999. pp. 100-110

A Mobile-Support-Station-Based Causal Multicast Algorithm in Mobile Computing Environment

CHAO-PING LI AND TING-LU HUANG†

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

(Received December 22, 1997; Accepted May 8, 1998)

ABSTRACT

Causal multicast is required for several distributed applications. In a mobile computing environment, it is especially important for applications that involve human interactions from several locations, for example, teleconferencing. In this paper, we present a causal multicast algorithm in which the message overhead is independent of the number of mobile hosts, and which can handle connections/disconnections easily. It also handles dynamically changing groups. The algorithm is suitable for a mobile computing environment.

Key Words: causal multicast, causal delivery, mobile computing, distributed algorithms

I. Introduction

Miniaturization of computers and the rapidly expanding technology of cellular communications has made it possible for mobile users to access information anywhere and anytime. These technologies come together in the form of mobile computing. Many of the distributed algorithms designed for distributed systems with fixed hosts only cannot be directly used in a mobile computing environment because of the change in physical network connectivity, the resource constraints of mobile hosts and the limited bandwidth of a wireless link. This has led to a considerable amount of research into adapting algorithms designed for conventional distributed systems to make them suitable for a mobile computing environment. In this paper, we consider the problem of providing an important communication support, causal multicasting messages, for mobile hosts.

Consider two messages m and m' sent to the same destination such that the sending of m "happens before" the sending of m'. Causal ordering of message delivery is obeyed if m is delivered before m' is delivered. Techniques for the causal ordering of messages are useful in developing distributed algorithms and may simplify the algorithms themselves (Birman and Joseph, 1987). Causal ordering has been regarded as an important building block for constructing reliable distributed systems (Jalote, 1994). In a mobile computing environment, causal ordering is especially important for applications that involve human interactions from several locations. Some of the major applications of distributed mobile systems in which causal ordering is useful are teleconferencing, stock trading, collaborative applications etc. (Alagar and Venkatesan, 1994; Prakash *et al*., 1997).

There are several algorithms that implement causal ordering for distributed systems with static hosts only (Birman and Joseph, 1987; Birman *et al*., 1991; Prakash *et al*., 1997; Raynal *et al*., 1991; Schiper *et al*., 1989), and there are some for a mobile computing environment (Alagar and Venkatesan, 1994; Prakash *et al*., 1997). To enforce causal ordering, the algorithms require that extra information be appended to each message, thus incurring a message space overhead. For each of the algorithms designed for stationary distributed systems, the message space overhead is at least $O(N^2)$, where N is the number of processes in the system. Hence, the protocols are not scalable and, thus, are not suitable for a mobile computing environment due to the limitations of the available bandwidth and energy consumption.

In a mobile computing environment, one of the algorithms proposed by Alagar and Venkatesan (1994) requires only $O(n_{MSS}^2)$ message overhead, where n_{MSS} is the number of mobile support stations

†To whom all correspondence should be addressed.

(MSSs). Nevertheless, the handoff procedure of the algorithm needs $O(n_{MSS})$ message exchanges to handle mobility. Yen *et al.* (1996) presented a compromise algorithm with $O(n_{MSS} \times n_{MH})$ message overhead but a less complicated handoff procedure, where n_{MH} is the number of mobile hosts (MHs). However, both of these algorithms assume that there is an underlying routing protocol (Bhagwat and Perkins, 1993; Ioannidis *et al.*, 1991) for routing a message from an MH to another MH. As far as multicasting is concerned, the functionality of multicast thus can be achieved only by means of multiple unicasts, which results in poor utilization of the network bandwidth (Acharya and Badrinath, 1993). A multicast protocol for a mobile environment has been presented by Acharya and Badrinath (1993). However, this protocol does not enforce causal ordering. The algorithm proposed by Prakash *et al.* (1997) adopts the multicast algorithm presented by Acharya and Badrinath (1993) and appends only direct dependence information to each message to enforce causal multicast. However, its overhead is $O(n_{MH}^2)$. In summary, the existing protocols are not scalable in a mobile environment. Hence, there is a need for an implementation of *scalable* causal multicast in a mobile environment.

The rest of the paper is organized as follows. Section II contains a description of the system model and a formal definition of causal multicast. The algorithm is presented in Section III and its proof of correctness in Section IV. Section V compares the performance of our algorithm with that of related works. Finally, a conclusion is drawn in Section VI.

II. System Model and Definition

1. System Model

We use the model proposed in Badrinath *et al.* (1993) as the underlying execution environment of our protocol (Fig. 1). It also has been considered in many discussions on algorithms for mobile computing environments (Alagar and Venkatesan, 1994; Badrinath *et al.*, 1993, 1994; Prakash *et al.*, 1997; Yen *et al.*, 1996). The system consists of a set of MHs and fixed hosts. An MH is a host that can move while retaining its connectivity to the network. A fixed host is a host whose location does not change with time. A fixed host can also be an MSS. An MSS has the necessary infrastructure to communicate directly with MHs. The number of MHs, denoted as n_{MH}, is large whereas that of MSSs, denoted as n_{MSS}, is relatively small. Thus, we assume that $n_{MH} >> n_{MSS}$.

The geographical area that an MSS covers is called

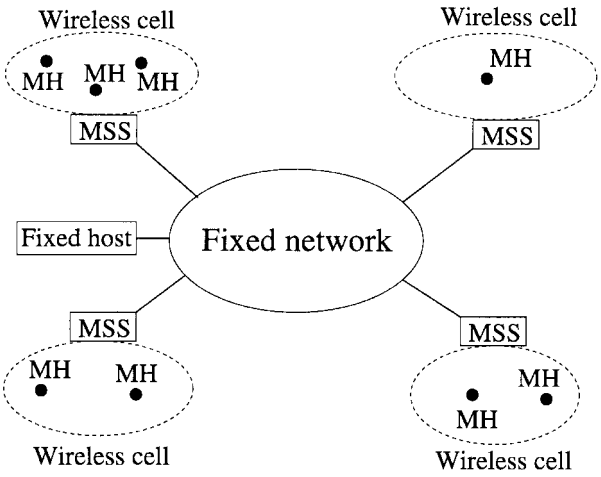

Fig. 1. Mobile environment architecture.

a cell. All MHs that are located within a cell of an MSS are considered to be local to the MSS. An MH may belong to at most one cell at any time. It can directly communicate with an MSS through a wireless channel only if it is local to the MSS. To send a message from one MH, MH_i, to another MH, MH_j, MH_i first sends the message to its local MSS over the wireless network. The MSS then forwards the message to the local MSS of MH_j, which will then forward the message to MH_j over the wireless network.

MSSs connect to one-another using wired channels. The MSSs and the wired channels constitute the static network. The overall network architecture thus consists of a *wired network* of fixed hosts and *low-bandwidth wireless networks*, each comprising a MSS and the MHs local to the cell. We assume that a logical channel exists between each pair of MSSs. The logical channel is assumed to be reliable but need not to be FIFO (First In First Out) whereas the wireless channel between an MSS and each of its MH is assumed to be reliable and FIFO.

When an MH moves from one cell to another, a handoff procedure is executed, whereby the algorithm-specific data structure on behalf of the MH maintained by the previous MSS is transferred to the new MSS. This procedure will be described in more detail in Subsection 2. In rare circumstances, it is possible for an MH to move again to another new cell before the handoff procedure is completed due to the combination of the small sized cells of MSSs and fast movements of MHs. In this paper, we will not discuss delivery of a message to an MH in such circumstances. Therefore, the system model requires that the static part of the network possesses sufficient host processing ability and communication speed, so that after an MH enters its present cell and before it subsequently leaves the

cell, the handoff procedure will have finished.

An MH 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 for any of the disconnected local MHs, the message can be stored and delivered to the MH after reconnection.

2. Physical Constraints and the Requirements

Mobile hosts have significantly less computing power compared to fixed hosts. In addition, an MH requires a stand-alone power resource (Badrinath *et al.*, 1993; Forman and Zahorjan, 1994). The fact that CPU operations, memory accesses, data transmission and reception all consume power leads us to believe that the number of operations executed by an MH should be minimal to reduce power consumption.

Additionally, wireless channels have significantly lower bandwidth than to those within fixed networks. Thus, the number of wireless messages should be kept to a minimum. All these considerations suggest that the relevant state information and data structure required for a protocol execution should be stored in MSSs.

To reduce power consumption further, MHs often operate in a doze mode: the host shuts down most of its functions and only listens for incoming messages (Badrinath *et al.*, 1993). On receiving a message, the MH resumes its normal mode of operation. The effectiveness of the doze mode will be mitigated if a significant portion of the protocol's state is maintained at an MH since control messages for updating the protocol's state will cause the MH to resume its normal mode. This suggests that protocols for MHs should be structured so that the communication and computation load of a protocol execution is placed on the static network as far as possible. Another operation mode to be taken into account is the disconnection mode. An MH may connect or disconnect itself frequently. Thus, we require that the protocols handle connections/disconnections effectively and gracefully.

3. Definition

An event in a mobile computing environment can be the sending of a message, the receiving of a message or an internal action. Since there is no perfectly synchronized clock, events in such an environment are ordered based on Lamport's happen before relation, $\rightarrow$, defined as follows (Lamport, 1978).

For any two events a and b, $a \rightarrow b$ is true if and only if

(1) a and b occur at the same host, and a occurs before b,

(2) a is a sending of a message, and b is the receipt of that message, or

(3) there exists an event c such that $a \rightarrow c$ and $c \rightarrow b$.

If $a \not\rightarrow b$ and $b \not\rightarrow a$, then a and b are said to be concurrent, denoted as $a \| b$.

Let $sent(m)$ be the event that corresponds to the sending of message m and $recv(m)$ be the event that corresponds to the receipt of message m. Causal ordering of message delivery (COMD) is obeyed if, for any two messages m_1 and m_2 that have the same destination, $sent(m_1) \rightarrow sent(m_2)$ then $recv(m_1) \rightarrow recv(m_2)$. Similarly, causal multicast (CM) is a multicast that obeys causal ordering of message delivery. When messages are sent using traditional transport protocols, COMD or CM might be violated due to the delay of the communication channel. Implementing COMD or CM means adding a protocol to the original system such that, as far as application is concerned, causal ordering is never violated. To avoid confusion, we will hereafter consistently use the term "receive" to denote a message received by the original system and "deliver" to denote a message delivered to the application level. That is, let deliv(m) denote the event in which message m is delivered; a COMD or CM algorithm can ensure that $sent(m_1) \rightarrow sent(m_2)$ always implies that $deliv(m_1) \rightarrow deliv(m_2)$.

III. Our Algorithm

This section will present the algorithm for *causal multicasting* of messages to MHs. An execution of the protocol may be requested either by an MH or a fixed host. *When it is requested by an MH, the local MSS is responsible for executing the protocol on behalf of the MH.* However, a non-MSS fixed host can also make such a request. We assume that the request is forwarded to an MSS which then executes the protocol. In the sequel, the term *initiator* denotes the MSS executing the protocol on behalf of an MH or another fixed host. For simplicity, we only consider multicast messages that are addressed to MHs, excluding those that are addressed to fixed hosts.

Our algorithm consists of *three* modules: the *WIRED* module, *HANDOFF* module and *WIRELESS* module. The three modules communicate with each other. The *WIRED* module, executed in a fixed network, is the main module for controlling causal multicast delivery. The *HANDOFF* module is executed when an MH moves from one cell to another, and is responsible for transferring relevant information to the new cell. The *WIRELESS* module is responsible for delivering

a message from/to MHs.

1. WIRED Module

The WIRED module is the main part of our algorithm for controlling causal multicast. We will describe the multicast assumption, the MSS-based causal multicast scheme, the data structure and the algorithm of the module in the following.

A. Multicast Assumption

Multicasting a message in a mobile environment can be achieved by means of multiple unicasts using point-to-point mobile internetworking schemes, such as those presented by Bhagwat and Perkins (1993) and Ioannidis *et al*. (1991). However, this results in an increase of the power consumption at the sender MH and poor utilization of both wired and wireless links (Acharya and Badrinath, 1993). Moreover, routing protocols in a network with MHs incurs either a search cost (Ioannidis *et al*., 1991) or an inform cost (Bhagwat and Perkins, 1993) in tracking the location of individual MHs.

The multicast part of our algorithm adopts the basic idea of the multicast protocol presented by Acharya and Badrinath (1993): when a multicast message is sent by MH_k, it is first sent to the local MSS of MH_k, and then the MSS sends the message to *all* MSSs. After receiving the message, the MSS forwards this message to its local MHs which are the destinations of the message and buffers it. When an MH moves to the cell, the MSS will check the buffer and forward the message in it to the MH if the message has not been delivered to the MH. Moreover, the MSS can delete the buffered message only after the message is delivered to all the destinations to guarantee at-least-once delivery.

B. The MSS-based Causal Ordering Scheme

We assume that the wireless channel between an MSS and an MH in its cell is FIFO. If the MHs never move, maintaining the causal multicast in the MSS level can also ensure causal multicast in the MH level (Alagar and Venkatesan, 1994). To ensure causal ordering among MHs, a message *m* only needs to carry information to enforce causal ordering of message delivery among MSSs. Hence, the message space overhead is greatly reduced. However, the mobility of the MHs may cause the CM to be violated (Alagar and Venkatesan, 1994). In addition, it may also cause the MHs to deliver a message more than once or fail to deliver a message (Acharya and Badrinath, 1993). The module is a two-level delivery protocol: before a message is delivered to an MH, the message has to be first delivered to the MSS to which the MH is local. The MSS, after being delivered, then forwards the message to the MH.

C. Data Structure

Each MSS_i, as a proxy, maintains an integer, seq_no_i, to count the number of messages it has *initiated* so far. The counter is set to *zero* at the beginning. Each time an MSS_i receives a multicast request, seq_no_i is incremented by one. For example, if seq_no_i equals x, this implies that MSS_i has initiated x multicast messages for MHs. MSS_i also maintains a *vector* of length n_{MSS}, called $DELIV_i$, to track the delivery information for MSS_i. Each entry of $DELIV_i$ is set to *zero*, i.e., (0, 0, ..., 0), initially. The vector records the number of messages initiated from other MSSs which have been delivered to MSS_i. For example, if $DELIV_i[j]$ equals x, where $i \neq j$, this denotes that all the messages sent by MSS_j with seq_no_j *less* than or *equal* to x have been delivered to MSS_i.

An MSS acts as a *proxy* for the MHs which are local to it and also maintains two vectors, $DELIV\_MH_k$ and $SENT\_MH_k$, of length n_{MSS} for each of its local MH_k. $DELIV\_MH_k$ denotes the greatest sequence number, on per-MSS basis, that has been delivered to MH_k. For example, if $DELIV\_MH_k[j]$ equals x, this implies that the greatest sequence number of all the messages initiated by MSS_j that have been delivered to MH_k is x. $SENT\_MH_k$ denotes the knowledge of MH_k about the number of messages that have been initiated from each MSS. For example, if $SENT\_MH_k[j]$ equals x, this means that x messages have been initiated from MSS_j by the knowledge of MH_k. The two vectors will be transferred to the new cell if the MH moves to a new cell.

When an MSS initiates a multicast, it sends the message with the control information appended to *all* MSSs. The control information consists of the *initiator* id, *destination* ids and *timestamp* of the message. Each MSS uses the control information received with the message to determine if the message can be delivered to the MSS or if it should be *buffered* until its causal predecessors meant for the MSS are delivered. If the message is delivered, it will be put in MSS_BUFER_i to terminate the *first* level (*MSS* level) delivery of the protocol. The *second* level (*MH* level) delivery of the protocol compares $DELIV\_MH_k$ with the timestamp of the message for each local destination MH_k to determine if the message has been delivered to the destination MH at other MSSs or not. If not, the message will be put in a deliver queue, $DELIV\_Q\_MH_k$, for MH_k. The WIRELESS module then forwards the messages in $DELIV\_Q\_MH_k$ one by one to MH_k.

D. The Algorithm

(1) On receiving a **causal multicast (CMcast) request** from an MH_k, the local MSS_i initiates the protocol as follows.
  (i) Before sending m, MSS_i executes the following steps.
    /* *It increments* seq_no_i *and timestamps m.* */
    (a) $seq\_no_i := seq\_no_i + 1$;
    (b) $SENT\_MH_k[i] := seq\_no_i$;
    (c) $timestamp(m) := SENT\_MH_k$;
  (ii) MSS_i sends *CMcast(i,m,timestamp(m),dests(m))* to *all* MSSs.

(2) On receiving a ***CMcast***() from MSS_j, MSS_i executes the steps below.
  (i) **Wait until**: /* *Beginning of the MSS level delivery* */
    (a) $timestamp(m)[j] = DELIV_i[j] + 1$ and,
    (b) $timestamp(m)[l] \le DELIV_i[l] \quad \forall l \in \{1 \ldots n_{MSS}\} - \{j\}$.
    /* *It's key to MSS level delivery. Step i(a) ensures that* MSS_i *has delivered all the messages initiated by* MSS_j *that precede m. Step i(b) ensures that* MSS_i *has delivered all those messages received by* MH_k *before* MH_k *makes the request.* */
  (ii) $DELIV_i$ is updated in the following manner.
    (a) $DELIV_i[j] := timestamp(m)[j]$.
  (iii) m is appended to $DELIV\_BUF_i$ along with $dests(m)$.
    /* *m can be deleted from the buffer only if* MSS_i *have received the Delete(m) message from the initiator. MSS level delivery is finished.* */
  (iv) $\forall MH_k$ such that $(MH_k \in M\_Local_i)$ and $(MH_k \in dests(m))$
    /* *Beginning of MH level deliver* */
    if $(timestamp(m) \le DELIV\_MH_k)$[1]
    then
      delete k from $dests(m)$;
    /* *The message has been delivered to* MH_k *at other MSSs.* */
    else
      insert message m into $DELIV\_Q\_MH_k$;

(3) The *WIRELESS* module of MSS_i sends the messages in $DELIV\_Q\_MH_k$ one by one via wireless media. When MH_k acknowledges the receipt of m, the *WIRED* module at MSS_i will be notified and will execute as follows:
  (i) delete m from $DELIV\_Q\_MH_k$;
  (ii) $\forall x \in 1 \ldots n_{MSS}$

$$\begin{cases} DELIV\_MH_k[x] = \max(DELIV\_MH_k[x], \\ \qquad\qquad timestamp(m)[x]) \\ SEND\_MH_k[x] = \max(SEND\_MH_k[x], \\ \qquad\qquad timestamp(m)[x]) \end{cases}$$

  (iii) send ***Ack***(***m***, ***k***) to the initiator of message m.

(4) On receiving ***Ack***(***m***, ***k***) from an MSS, the initiator executes as follows:
  (i) delete k from $dests(m)$;
  (ii) **if** $dests(m) = \varnothing$ **then**
    send $Delete(m)$ to all MSSs;

(5) On receiving a ***Delete***(***m***) message from the *initiator* of m, MSS_i executes as follows:
  (i) delete m and its related information in the $DELIV\_BUF_i$.
    /* *Message m has been delivered to all destination MHs, so that MSSs no longer need to maintain it.* */

2. HANDOFF Module

The protocol relies on this module to transfer an MH's state to its current cell when the MH moves to a new cell. In our model, we assume that each MSS periodically broadcasts a beacon so that an MH can discover its transfer to a new cell. The module is activated when an MH discovers that it has moved to a new cell. The module uses the *handoff* procedure presented by Acharya and Badrinath (1993) and described in the following. Consider MH_k that moves from the cell under MSS_M to MSS_N. On discovering that it has moved to a new cell, MH_k sends a $greeting(k,M)$ message with its own identity k and the id of its previous MSS as the parameters to MSS_N. On receiving $greeting(k,M)$, MSS_N sends $deregister(k)$ to MSS_M. MSS_M deletes MH_k from MSS_local_M after receiving the *deregister* message from MSS_N and then transfers the relevant information about MH_k to MSS_N via a $register(k,data)$ message. In response, MSS_N adds k to the list MSS_local_N and stores the data about MH_k at MSS_N. The lists, MSS_local_N and MSS_local_M, contain ids of the MHs local to the cells of MSS_N and MSS_M, respectively.

The *handoff* procedure is completed when MSS_N has received the *register* message from MSS_M. From

[1] The relations between two vectors, a and b, are defined as follows:
$a \le b \Leftrightarrow \forall i,\ a[i] \le b[i]$,
$a < b \Leftrightarrow a \le b$ and $a \ne b$.

then on, MSS_N takes over the jobs of previous MSS for MH_k. If MH_k migrates to another MSS, $MSS_{N'}$, before the current *handoff* procedure is completed, the new migration message issued by $MSS_{N'}$ will not be handled until the current *handoff* procedure is completed.

In addition to transferring the algorithm related data structure to the new cell, the HANDOFF module also needs to ensure that the FIFO channel between the MH and its previous MSS is flushed properly (Acharya and Badrinath, 1993). To flush the channel properly, an MH_k maintains an integer number, $seq_{M\_to\_k}$, to record the sequence number of the last message received on the FIFO channel from its local MSS to the MH. Similarly, an MSS maintains an integer variable, $seq_{k\_to\_M}$, for each local wireless channel to record the sequence number of the last message received in sequence from each local MH_k to the MSS.

Let an MH_k move from the cell of MSS_i to that of MSS_j. The module is executed as follows.

(1) On receiving a ***deregister***(***k***, ***seq***$_{M\_to\_k}$) message from MSS_j, MSS_i begins execution as follows:
  (i) MH_k is deleted from MSS_Local_i;
  (ii) use $seq_{M\_to\_k}$ as an implicit acknowledgment;
  (iii) notify the WIRED module to update $DELIV\_MH_k$ and $SENT\_MH_k$;
  (iv) wait for $DELIV\_MH_k$ and $SENT\_MH_k$ from WIRED module;
  (v) Send *Register*(k, $DELIV\_MH_k$, $SENT\_MH_k$, $seq_{k\_to\_M}$) to the HANDOFF module at MSS_j;
  (vi) Delete the information relevant to MH_k.

(2) MSS_j, on receiving a ***register***() message, begins to execute the *HANDOFF* module as follows:
  (i) Store $DELIV\_MH_k$ and $SENT\_MH_k$ at MSS_j;
  (ii) Create $DELIV\_Q\_MH_k$ for MH_k:
    Add k to MSS_Local_j;
    $\forall m \in DELIV\_BUF_j$ such that ($MH_k \in dests(m)$)
    if ($timestamp(m) \le DELIV\_MH_k$) then
      delete k from $dests(m)$;
    else
      put m in $DELIV\_Q\_MH_k$;
  (iii) Send *Arrival*(k, $seq_{k\_to\_M}$) to the WIRELESS module at MSS_j;
    /**The WIRELESS module can use $seq_{k\_to\_M}$ to find out if the causal multicast request has been received by the previous MSS.* */

3. WIRELESS Module

The WIRELESS module is the only module at an MSS_i that can communicate with its local MHs via wireless media. The interaction between the WIRELESS module of an MSS_i and a local MH_k is described below:

(1) MH_k, upon discovering that it has **entered a new cell**, executes as follows:
  (i) Send *greeting*(k, $seq_{M\_to\_k}$) message to the WIRELESS module of the new cell.

(2) MSS_i, on receiving an ***arrival***(***k***, ***seq***$_{k\_to\_M}$), executes as follows:
  (i) Send *init*($seq_{k\_to\_M}$) to MH_k;
  (ii) Set $seq_{k\_to\_M}$ to *zero*;
  (iii) it then continues to execute the following loop.

```
While (h ∈ MSS_Local_i) do {
    if ¬(DELIV_Q_MH_k = ∅) then
        Deliver message m at the top of the queue;
    if an Ack is received from MH_k then
        Notify the WIRED module to update
        DELIV_MH_k and SENT_MH_k;
    if a causal multicast request is received from
        MH_k then {
        increment seq_{k_to_M} by one;
        notify the WIRED module;
        }
}
```

(3) On receiving an ***init***(***seq***$_{k\_to\_M}$), MH_k executes as follows:
  (i) use $seq_{k\_to\_M}$ as an implicit acknowledgment;
  (ii) initialize $seq_{M\_to\_h}$ to zero;
  (iii) retransmit the lost causal multicast request.

IV. Correctness Proof

We will prove the correctness of the protocol in three stages. First, we will show that causality is never violated (safety property), and we will then demonstrate that it never delays a message indefinitely (liveness property). Last, we show the exactly once delivery property. However, before developing these proofs, we need some preliminary results that are given in Subsection 1. Then, in Subsection 2, we will prove the safety property of our algorithm. In Subsection 3, we will prove the liveness property of our algorithm. Finally, in Subsection 4, we will prove the exactly once delivery property of our algorithm.

1. Preliminary Results

To prove the correctness of our protocol, we need some preliminary results.

Observation 1. Consider a multicast message m ini-

tiated by MSS_i such that $timestamp(m)[j]>0$, where $j\neq i$, and let $timestamp(m)[j]=x$. Then, m_x, the xth message initiated from MSS_j, should have been sent.

Observation 2. Let m be the xth message sent by MSS_i. Then, $timestamp(m)[i]=x$.

Observation 3. Consider two messages m_1 and m_2 such that $sent(m_1)\rightarrow sent(m_2)$ then $timestamp(m_1)< timestamp(m_2)$.

Observation 4. For two messages m_1 and m_2, let $sent(m_1)\rightarrow sent(m_2)$, and let both of them be initiated by the same MSS, say MSS_i. Then, $timestamp(m_1)[i]< timestamp(m_2)[i]$.

Lemma 1. Consider a message m initiated by MSS_i. If message m has not been delivered to MSS_j, then $DELIV_j[i]<timestamp(m)[i]$.

Proof. Let m be the xth message from MSS_i, where $timestamp(m)[i]=x$, $DELIV_j[i]<x$, and assume that m has not been delivered to MSS_j. In the following, we will prove it by contradition. Suppose that there exists a message m' ($m\neq m'$) with $timestamp(m')[i]\geq x$ that can be delivered to MSS_j before message m so that it can make $DELIV_j[i]\geq x$ without delivering message m. There are two possible conditions:

Case 1. m' is sent from MSS_i; then, $timestamp(m')[i] >x$ (because $m\neq m'$). $timestamp(m')[i]\neq DELIV_j[i]+1$. It does not satisfy the first condition of Step(2) of the WIRED module and, thus, is delayed.

Case 2. m' is sent from MSS_l, where $l\neq i$. We have $timestamp(m')[i]>DELIV_j[i]$; similarly, it does not satisfy the second condition of Step(2) of WIRED module and, thus, is also delayed.

Thus, this assumption leads to a contradiction. □

Corollary 1. If a message m is initiated from MSS_i, the message m' with $timestamp(m')[i]\geq timestamp(m)[i]$ will be delivered after m is delivered.

Lemma 2. Consider a message m initiated from MSS_i, where MH_k is one of its destinations. If message m has not been delivered to MH_k (MH level deliver), then $DELIV\_MH_k[i]<timestamp(m)[i]$.

Proof. In the following, we will prove it by contradiction. Let m be the xth message from MSS_i, where $timestamp(m)[i]=x$ and $DELIV\_MH_k[i]<x$. Suppose that there exists a message m' ($m'\neq m$) with $timestamp(m')[i]\geq x$, and assume that the message is delivered to MH_k before message m.

By Corollary 1, message m will be delivered before m' in the MSS level. Since the wireless channels are FIFO, MH_k delivers m before m', thus leading to a contradiction. □

2. Safety Property Proof

Suppose two messages m_1 and m_2, such that $SEND(m_1)\rightarrow SEND(m_2)$, are both destined for MH_k. As far as MH_k is concerned, we will prove that m_2 can not be delivered to MH_k if m_1 has not been delivered. Consider the actions of MSS_i, which receives the two messages.

Case 1. m_1 and m_2 are both initiated by the same MSS, say MSS_j. By Observation 4, we have

$$timestamp(m_1)[j]<timestamp(m_2)[j]. \quad \textbf{(1)}$$

Furthermore, by Lemma 1, if m_1 has not been delivered to MSS_i, we have

$$DELIV_i[j]<timestamp(m_1)[j]. \quad \textbf{(2)}$$

From **(1)** and **(2)**, $DELIV_i[j]\neq timestamp(m_2)$, and m_2 is, thus, delayed by Step(2) of the WIRED module.

Case 2. m_1 and m_2 are initiated by two different MSSs, say MSS_j and $MSS_{j'}$, separately. We will show that m_2 can not be delivered before m_1 by induction on the number of messages delivered to MSS_i.

Observe first that $send(m_1)\rightarrow send(m_2)$ by Observation 3; thus, we have

$$timestamp(m_1)<timestamp(m_2).$$

In particular, consider the field corresponding to MSS_j, the initiator of m; we have

$$timestamp(m_1)[j]\leq timestamp(m_2)[j]. \quad \textbf{(3)}$$

Base step. The first message delivered to MSS_i can not be m_2.

Recall that if no message has been delivered to MSS_i, then $DELIV_i[j]=0$. However, $timestamp(m_1)[j]>0$ (because m_1 is sent by MSS_j); hence, $timestamp(m_2)[j]>0$ and m_2 can not be the first message delivered to MSS_i.

Induction steps. Suppose that MSS_i has delivered k messages, none of which is a message m such that $SEND(m_1) \rightarrow SEND(m)$. If m_1 has not been delivered, then by Lemma 1 we have

$$DELIV_i[j] < timestamp(m_1)[j]. \quad \textbf{(4)}$$

From **(3)** and **(4)**, it follows that

$$DELIV_i[j] < timestamp(m_2)[j].$$

Again, by Step(2) of the WIRED module, m_2 can not be the k+1st message delivered to MSS_i.

3. Liveness Property Proof

In the following, we will prove that our protocol never violates the liveness property; i.e., our protocol will not delay a message indefinitely.

Suppose that there exists a multicast message m initiated by MSS_j that cannot be delivered to MSS_i indefinitely. Step(2) of the WIRED module implies that either

$$timestamp(m)[j] \neq DELIV_i[j]+1 \text{ or}$$

$$\exists l \ni timestamp(m)[l] > DELIV_i[l], \text{ where } l \neq j.$$

We will consider two cases in turn.

Case 1. $timestamp(m)[j] \neq DELIV_i[j]+1$. This means that m is not the next message to be delivered to MSS_i from MSS_j. There are only a finite number of messages from MSS_j that can precede m. Since messages are sent to all MSSs and the channels between MSSs are FIFO, it follows that there must be some message m' sent by MSS_j that MSS_i has received but not delivered, and that it is the next message from MSS_j to be delivered, i.e., $timestamp(m)[j] = DELIV_i[j]+1$. If m' is also delayed, this delay must be caused by case 2.

Case 2. $\exists l \ni timestamp(m)[l] > DELIV_i[l]$, where $l \neq j$. Let $timestamp(m)[l] = x$. By Observation 1, the xth message m' sent from MSS_l should has been sent before m was sent. Message m_x either has not yet been received or has been received but is delayed by MSS_i. However, since messages are broadcast in the MSS level, m' is received eventually by MSS_i. The same reasoning applied to m can also be applied to m'.

The number of messages that must be delivered before m is finite, thus leading to a contradiction. This means that our protocol never delays a message indefinitely. □

Table 1. Comparison between Related Works

Algorithms	Message Space Overhead	Memory Overhead
Our algorithm	$O(n_{MSS})$	$O(n_{MSS})$
AV-94-1	$O(n_{MH}^2)$	$O(n_{MH}^2)$
AV-94-2	$O(n_{MSS}^2)$	$O(1)$
AV-94-3	$O(n_{MSS}^2 \times k^2)$	$O(n_{MSS}^2 \times k^2)$
YHH-96	$O(n_{MSS} \times n_{MH})$	$O(n_{MSS} \times n_{MH})$
PRS-95	$O(n_{MH}^2)$	$O(n_{MH}^2)$

Notes: n_{MSS}: number of MSSs.
n_{MH}: number of MHs.
n_{dst}: number of destination MHs for a certain multicast message.
k: number of logical MSS for a physical MSS.

Table 2. Comparison between Related Works (Cont.)

Algorithms	Handoff Complexity		Possibility of Inhibition
	number of messages	message size	
Our algorithm	$O(1)$	$O(n_{MSS})$	Yes
AV-94-1	$O(1)$	$O(n_{MH}^2)$	No
AV-94-2	$O(n_{MSS})$	$O(1)$	Yes
AV-94-3	$O(n_{MSS} \times k)$	$O(1)$	Yes
YHH-96	$O(1)$	$O(n_{MSS} \times n_{MH})$	Yes
PRS-95	$O(1)$	$O(n_{MSS}^2)$	No

4. Exactly Once Property Proof

In this section, we will prove that the destination of a message receives exactly one copy of the message. Consider a message m, and assume that MH_k is one of the destinations of the message. There are two possible situations for MH_k to deliver the message.

Case 1. MH_k has received one copy of the message. By Step(3) of the WIRED module, $DELIV\_MH_k$ should have been updated to greater than $timestamp(m)$. Step(2) of the WIRED module makes it impossible to deliver another copy of the message.

Case 2. MH_k has not received the message at all. By Lemma 2, $DELIV\_MH_k[i] < timestamp(m)[i]$ and, thus, $DELIV\_MH_k \not\geq timestamp(m)$. Without loss of generality, we consider that MH_k moves to a new cell, MSS_i. There are also two possible situations:

Subcase 1. MSS_i has delivered message m. Since m has not been delivered to all of its destinations,

message *m* will still be buffered in $DELIV\_BUF_i$. Message *m* will be forwarded to $DELIV\_Q\_MH_k$ by Step(2) of the HANDOFF module.

Subcase 2. MSS_i has not delivered message *m*. Since our algorithm satisfies the liveness property, MSS_i will eventually deliver message *m*. By Step(2) of the WIRED module, message *m* will be forwarded to $DELIV\_Q\_MH_k$.

V. Analysis and Comparison

In this section, we will analyze our algorithm and compare it with related works. The comparison results are shown in Tables 1 and 2, and more detailed discussion will be given in the following sections. Section V.l describes the message space overhead. Section V.2 contains a description about the memory overhead. Handoff complexity analysis is presented in Section V.3. Finally, we will discuss the possibility of inhibition in Section V.4.

In the following, we will denote the Prakash-Raynal-Sinhal Algorithm (Prakash *et al*., 1997), Alagar-Venkatesan Algorithm (Alagar and Venkatesan, 1994) and Yen-Huang-Hwang Algorithm (Yen *et al*., 1996) as PRS-95, AV-94 and YHH-96, respectively, for abbreviation. Further, we will use AV-94-1, AV-94-2 and AV-94-3 to denote the first, second and the third algorithm of AV-94, respectively.

1. The Message Space Overhead

To maintain causal multicast, we need to append extra information to each message. The information is essentially overhead that increases the transmission delay when it is passed over the network. The larger the size of the information, the longer the delay of the transmission. In our algorithm, we append only a n_{MSS}-integer vector to each message to maintain causal multicast. Thus, the message space overhead of our algorithm is $O(n_{MSS})$. Typically, the MSSs constitute a small subset of all the nodes in a mobile computing system.

Consider the related works. The message space overhead is $O(n_{MH}^2)$ in PRS-95, $O(n_{MH}^2)$ in AV-94-1, $O(n_{MSS2}^2)$ in AV-94-2, $O(k^2 \times n_{MSS}^2)$ in AV-94-3 and $O(n_{MSS} \times n_{MH})$ in YHH-96. From the discussion above, it can be observed that our algorithm outperforms the related works with respect to the message space overhead.

2. Memory Overhead

In a mobile computing environment, to satisfy the basic requirements described in Section II.2, an MSS acts as a proxy and maintains the algorithm related data structure for its local MHs. We require that the amount of data be as small as possible. Recall that, as noted in Section III.l.C, an MSS maintains two vectors, *SENT_MH* and *DELIV_MH*, for each local MH. The memory overhead for each MH is $O(n_{MSS})$.

Consider the related works. The overhead is $O(n_{MH}^2)$ in PRS-95, $O(n_{MH}^2)$ in AV-94-1, $O(1)$ in AV-94-2, $O(k^2 \times n_{MSS}^2)$ in AV-94-3 and $O(n_{MH} \times n_{MSS})$ in YHH-96. It is worth noting that in AV-94-2, instead of storing the message for each MH, the MSS maintains the information for each logical MSS.

The memory overhead of our algorithm is less than that of any of the algorithms except AV-94-2. However, the handoff procedure of the algorithm in AV-94-2 requires $O(n_{MSS})$ message exchanges. Another important feature of our algorithm is that, unlike algorithm PRS-95, AV-94-1 and YHH-96, the data structure maintained by an MSS for its local MHs in our algorithm is independent of the number of MHs. Thus, MHs connections/disconnections do not affect the structure of the algorithm. Our algorithm, thus, can handle the two operations easily and is suitable for a mobile computing environment.

3. Handoff Complexity

When a MH moves from one cell to another, the handoff procedure is incurred to transfer the algorithm related data structure to its new cell. Recall that as explained in Section III.2, the HANDOFF module of our algorithm requires transferring of two vectors, *SENT_MH* and *DELIV_MH*, and uses $O(1)$ messages. Thus, our algorithm requires $O(1)$ messages of size $O(n_{MSS})$ when an MH moves to its new cell.

Consider the related works. Algorithm PRS-95 uses $O(1)$ messages of size $O(n_{MH}^2)$. The three algorithms in AV-94 use $O(1)$ messages of size $O(n_{MH}^2)$, $O(n_{MSS})$ messages of size $O(1)$ and $O(k \times n_{MSS})$ messages of size $O(1)$, respectively. YHH-96 requires $O(1)$ messages of size $O(n_{MSS} \times n_{MH})$. It is worth noting that the number of messages dominates the handoff complexity (Lazowska *et al*., 1986). Thus, the performance of $O(1)$ messages of size $O(n_{MSS})$ is better than that of $O(n_{MSS})$ messages of size $O(1)$.

From the comparison discussed above, our algorithm has better performance than the related works in terms of handoff complexity.

4. Possibility of Unnecessary Inhibition

The drawback of our algorithm is the possibility of a message being inhibited from being delivered to

an MH; i.e., it is possible that delivery of a message may be delayed even though delivery of the message does not violate causal ordering. This is because in our algorithm, causal ordering is maintained among MSSs. Delivery of a message may violate causal ordering from an MSS's point of view whereas its delivery to an MH may not violate causal ordering from the MH's point of view. Algorithms AV-94-2, AV-94-3 and YHH-96 also face this problem. However, all of the algorithms benefit from reduction in message space overhead.

In contrast, algorithm PRS-95 and AV-94-1 maintain causal ordering explicitly among MHs. Hence, inhibition never occurs. Both algorithms require $O(n_{MH}^2)$ message overhead, which can also cause transmission delay. A related simulation described by Alagar and Venkatesan (1994) shows that such inhibition is not serious: when n_{MH} is large enough, say 30, the disadvantage of inhibition will be cancelled out by the benefit gained from the reduction in message overhead since in such a situation, the message space overhead will dominate the delay in the system.

VI. Conclusions

In this paper, we have presented a causal multicast algorithm for a mobile computing environment. In such an environment, many physical constraints which do not exist in traditional distributed systems should be taken into consideration: low bandwidth of the wireless links, tight constraints on power consumption and significantly lower computing power. In the design of an algorithm which will satisfy these constraints, each MSS in the system acts as a proxy and maintains the algorithm related data structure for its local MHs in order to execute the protocol. Instead of maintaining causality explicitly among MHs, our algorithm enforces causal ordering only among MSSs. Thus, the message space overhead required to maintain causal ordering is proportional to the number of MSSs. Since the number of MSSs is much smaller than that of MHs, the overhead is reduced greatly. The MSS-based algorithm has been proved to satisfy the safety, liveness and exactly once delivery properties. It can also handle dynamically changing multicast communication groups.

The low memory and communication overhead satisfy the low energy consumption and low available bandwidth constraints of a mobile computing environment. The algorithm is, thus, easily scalable. In addition, the data structure maintained for each MH in order to enforce multicast is independent of the number of MHs. Connections/disconnection, thus, can be handled gracefully without modifying the data structure for each MH. Hence, the MSS-based algorithm is suitable for a mobile computing environment.

Acknowledgment

This research was supported by the National Science Council, R.O.C., under grant NSC 85-2213-E009-061.

References

Acharya, A. and B. R. Badrinath (1993) Delivering multicast messages in networks with mobile hosts. *Proceedings of the 13th International Conference on Distributed Computing Systems*, pp. 292-299. Pittsburgh, PA, U.S.A.

Alagar, S. and S. Venkatesan (1994) Causally ordered message delivery in mobile system. *Proceedings of Workshop on Mobile Computing Systems and Applications*, pp. 169-174. Santa Cruz, CA, U.S.A.

Badrinath, B. R., A. Acharya, and T. Imielinski (1993) Impact of mobility on distributed computations. *Operating Systems Review*, **27**(2), 15-20.

Badrinath, B. R., A. Acharya, and T. Imielinski (1994) Structuring distributed algorithms for mobile hosts. *Proceedings of the 14th International Conference on Distributed Computing Systems*, pp. 21-28. Pozman, Poland.

Bhagwat, P. and C. E. Perkins (1993) A mobile networking system based on internet protocol(ip). USENIX Symposium on Mobile and Location-Independent Computing. Cambridge, MA, U.S.A.

Birman, K. and T. Joseph (1987) Reliable communication in the presence of failures. *ACM Transactions on Computer Systems*, **5**(1), 47-76.

Birman, K., A. Schiper, and P. Stephenson (1991) Lightweight causal and atomic group multicast. *ACM Transactions on Computer Systems*, **9**(3), 272-314.

Forman, G. H. and J. Zahorjan (1994) The challenges of mobile computing. *IEEE Computers*, **27**(4), 38-47.

Ioannidis, J., D. Duchamp, and G. Q. Maruire, Jr. (1991) Ip-based protocols for mobile internetworking. *Proceedings of ACM SIGCOMM Symposium on Communication Architecture and Protocols*, pp. 235-245. Zurich, Switzerland.

Jalote, P. (1994) *Fault Tolerance in Distributed Systems*, Chap. 4, pp. 141-183. Prentice-Hall, Englewood Cliffs, New Jersey, U.S.A.

Lamport, L. (1978) Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, **21**(7), 538-565.

Lazowska, E. D., J. Zahorjan, D. R. Cheriton, and W. Zwaenepoel (1986) File access performance of diskless workstation. *ACM Transactions on Computer Systems*, **4**(3), 238-268.

Prakash, R., M. Raynal, and M. Singhal (1997) An adaptive causal ordering algorithm suited to mobile computing environments. *Journal of Parallel and Distributed Computing*, **41**, 190-204.

Raynal, M., A. Schiper, and S. Toueg (1991) The causal ordering abstraction and a simple way to implement it. *Information Processing Letters*, **39**, 343-350.

Schiper, A., J. Eggli, and A. Sandoz (1989) A new algorithm to implement causal ordering. *Proceedings of the 3rd International Workshop on Distributed Algorithms*. Nice, France.

Yen, L. H., T. L. Huang, and S. Y. Hwang (1996) A protocol for causally ordered message delivery in mobile computing systems. *Proceedings of the Second International Mobile Computing Conference*, pp. 35-41. Hsinchu, Taiwan, R.O.C.

在行動環境下以行動支援主機為主的多點因果遞送演算法

李照平　黃廷祿

國立交通大學資訊工程研究所

摘　要

對於許多分散式應用而言，多點因果遞送（Causal Multicast）是必要的。在行動計算（Mobile Computing）的環境中，它在牽涉到人類在各個不同地點間互動的應用中，如遠距會議（Teleconference）等，尤其重要。本論文中，我們提出了一個訊息空間負擔（Message Overhead）與行動主機（Mobile Host）之個數無關，而且容易處理行動主機斷線、連線之多點因果遞送演算法，它亦能處理動態改變的群組。這個演算法因此適合於行動計算環境。