A Lightweight Algorithm for Causal Message Ordering in Mobile Computing Systems

Chakarat Skawratananond, *Neeraj Mittal, and Vijay K. Garg

Electrical and Computer Engineering Dept.
The University of Texas at Austin
Austin, TX 78712

*Computer Science Dept.
The University of Texas at Austin
Austin, TX 78712

Abstract

Causally ordered message delivery is a required property for several distributed applications particularly those that involve human interactions (such as teleconferencing and collaborative work). In this paper, we present an efficient protocol for causal ordering in mobile computing systems. This protocol requires minimal resources on mobile hosts and wireless links. The proposed protocol is scalable and can easily handle dynamic change in the number of participating mobile hosts in the system. Our protocol, when compared to previous proposals, offers a low unnecessary delay, low message overhead and optimized handoff cost.

1 Introduction

The emergence of mobile computing devices, such as notebook computers and personal digital assistants with communication capabilities, has had a significant impact on distributed computing. These devices provide users the freedom to move anywhere under the service area while retaining network connection. However, mobile computing devices have limited resources compared to stationary machines. Distributed algorithms that run on the system with mobile computing devices therefore require some modifications to compensate for these factors.

In this paper, we consider causal message ordering required in many distributed applications such as management of replicated data [5, 6], distributed monitoring [4], resource allocation [11], distributed shared memory [2], multimedia systems [1], and collaborative work [12]. The protocols to implement causal message ordering in systems with static hosts have been presented in [7, 6, 9, 11, 13, 14]. These protocols can be executed by every mobile host with all the relevant data structures being stored on the mobile hosts themselves. However, considering limited resources and bandwidth of wireless links available to mobile hosts, it is not appropriate to apply these protocols directly to mobile systems. As introduced in [3], the following factors should be taken into account in designing protocols for mobile systems. Computation load on mobile hosts, and communication overhead in the wireless medium should be minimal. Also, protocols should be scalable, and be able to easily handle the effect of hosts connections and disconnections.

While ordering of messages in distributed systems with static hosts has received wide attention, there has been little work on causal message ordering in mobile computing systems. Alagar and Venkatesan [3] proposed three algorithms based on the algorithm by Raynal, Schiper and Toueg (RST) in [11]. The first algorithm ($AV1$) maintains causal ordering among all mobile hosts (MHs). The message overhead is proportional to the square of the number of MHs (n_h). However, the data structures required in the algorithm are stored in mobile support stations (MSSs) to reduce load on mobile hosts and wireless links. In the second algorithm ($AV2$), causal ordering is exclusively maintained among MSSs. The message overhead reduces to the square of the number of MSSs (n_s). Since stronger ordering is imposed, messages may experience unnecessarily delay even though they do not violate causal ordering in the mobile hosts' view. Their third algorithm ($AV3$) is aimed at reducing this unnecessary delay by partitioning each physical MSS into k logical support stations. As k increases, the degree of unnecessary delay decreases, but the message overhead and the cost of handling host migration increases.

Yen, Huang, and Hwang (YHH) [16] proposed another algorithm based on [11]. The message overhead in their algorithm lies between that of $AV1$ and $AV2$. In particular, each MSS maintains a matrix of size $n_s \times n_h$. The unnecessary delay in their algorithm is lower than $AV2$. Their handoff module is also more efficient than $AV2$. Prakash, Raynal, and Singhal (PSR) [10] presented an algorithm where message overhead is relatively low; however, in the worst case, it can be as large as $O(n_h^2)$.

In this paper we propose a new protocol in which message overhead structure is independent of the number of hosts in the system. As a result, our protocol is scalable and suitable for the systems where the number of participating hosts is varied dynamically. Our contribution can be summarized as follows: (1) With our protocol, we are able to decrease the unnecessary delivery delay while maintaining low message overhead. (2) Our handoff module is more efficient than $AV2$ and $AV3$ because we do not require the messages exchanged among mobile support stations to be causally ordered. (3) We discovered that *YHH* does not satisfy the liveness property. (4) Finally, we state and prove the condition implemented by our static module. We also present conditions implemented by $AV2$ and *YHH* (corrected) algorithms.

The rest of the paper is organized as follows. Section 2 presents the system model and the notation used in the paper. Sufficient conditions for causal message ordering in mobile computing systems are presented in Section 3. We present our protocol in Section 4. We compare our protocol with the previous work in Section 5. The simulation results are presented in Section 6. Section 7 concludes the paper.

2 System Model and Definitions

A mobile computing system consists of two kinds of processing units: *mobile hosts*, and *mobile support stations*. A mobile host (MH) is a host that can move while retaining its network connections. A mobile support station (MSS) is a machine that can communicate directly with mobile hosts over *wireless* channels. The geographical area within which an MSS supports MHs is called a *cell*. Even though cells may physically overlap, an MH can be directly connected through a wireless channel to at most one MSS at any given time. An MH can communicate with other MHs and MSSs only through the MSS to which it is directly connected. We assume that the wireless channels are FIFO, and both wired and wireless channels are reliable and take an arbitrary but finite amount of time to deliver messages. A mobile host can disconnect itself from the network and can reconnect at a later time.

Let $\mathcal{H} = \{h_1, h_2, \ldots, h_{n_h}\}$ represent the set of mobile hosts and $\mathcal{S} = \{S_1, S_2, \ldots, S_{n_s}\}$ denote the set of mobile support stations. In general, $n_h \gg n_s$. A mobile computation can be illustrated using a graphical representation referred to as *concrete diagram*. Figure 1 illustrates such a diagram where the horizontal lines represent MH and MSS processes. h_s is in the cell of S_i. h_d is in the cell of S_j. A concrete diagram in which only MH processes are shown is referred to as an *abstract diagram*.

An *application* message is a message sent by an MH intended for another MH. Since mobile hosts do not communicate with each other directly, an MH, say h_s, first sends an application message m to its MSS, say S_i, which then forwards m to the MSS, S_j, of the destination host, h_d. Figure 1 illustrates our notation. We define the *delivery* event as a local event that represents the delivery of a received message to the application or applications running on that process.

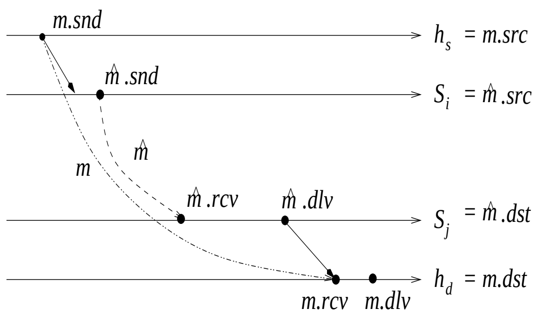

Figure 1: Notation

For any two events e and f on some mobile host, we write $e \prec_h f$ iff e occurs before f in real-time. Similarly, $e \prec_s f$ iff e occurs before f in real-time on some mobile support station. We use $\rightarrow_h$ and $\rightarrow_s$ to denote the Lamport's *happened before* relation [8] in the abstract and concrete diagram respectively. For any pair of messages m_i and m_j, we say that m_i *causally precedes* m_j in abstract view, denoted by $m_i \rightarrow_h m_j$, iff $m_i.snd \rightarrow_h m_j.snd$. Also, m_i causally precedes m_j in concrete view, denoted by $\hat{m}_i \rightarrow_s \hat{m}_j$, iff $\hat{m}_i.snd \rightarrow_s \hat{m}_j.snd$. A mobile computation is causally ordered if the following property is satisfied for any pair of application messages, m_i and m_j, in the mobile system,

$$(\mathcal{CO}) \qquad m_i.snd \rightarrow_h m_j.snd \;\Rightarrow\; \neg(m_j.dlv \prec_h m_i.dlv)$$

3 Sufficient Conditions

Theorem 1 *: A mobile computation with static MHs is causally ordered if (C_1) all wireless channels are FIFO, (C_2) messages in the wired network are causally* *ordered, and (C_3) each MSS sends out messages in the order they are received.*

The proof is provided in [15]. Condition C_2 can be formally expressed as

$$(\mathcal{CO}') \quad \hat{m}_i.snd \rightarrow_s \hat{m}_j.snd \Rightarrow \neg(\hat{m}_j.dlv \prec_s \hat{m}_i.dlv)$$

Sufficient conditions given in Theorem 1 were implicitly used in [3]. For systems with multiple static hosts, Theorem 1 gives a lightweight protocol for causal message ordering. In the extreme case when the entire computation is in a single cell, causal ordering can be provided by simply using FIFO between MHs and the MSS. However, C_1, C_2, and C_3 are not necessary. This is because we can construct a causally ordered computation such that C_1 and C_2 do not hold.

4 Algorithm

Alagar and Venkatesan extended RST [11] to mobile systems. In $AV2$, causal message ordering is maintained among MSSs. All MHs in a cell share a single matrix. The message and storage overhead is reduced to $O(n_s^2)$. This however can create false causal dependencies between messages. Figure 2 displays a sample of false dependencies created by $AV2$. In order to reduce these false causal dependencies and hence the unnecessary delay in $AV2$, we propose to use a separate matrix for each MH in a cell. The next two subsections describe the static and the handoff modules of our protocol. The static module is executed when an MH is in a particular cell. The handoff module is executed when an MH moves from one cell to another.

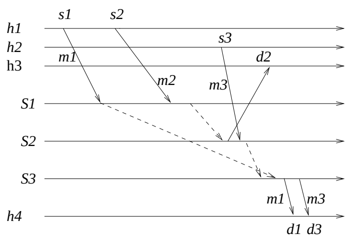

Figure 2: An example of unnecessary delay in $AV2$.

4.1 Static Module

For convenience, we first describe the static module assuming that hosts do not move. In the next subsection, we describe the handoff module and the modifications that need to be made to the static module to incorporate mobile hosts.

Our static module is based on RST. For simple exposition of the protocol, we assume that the channels among the MSSs are FIFO. This assumption can be easily relaxed by implementing FIFO among MSSs using sequence numbers. We also assume that every MSS knows about the location of the MHs. For each MH h_l, we maintain an $n_s \times n_s$ matrix M_l. $M_l[i,j]$ denotes the total number of messages h_l knows to have been sent by S_i to S_j. Assume that h_l is in the cell of S_i. In order to reduce the communication and computation overhead of h_l, the matrix M_l is stored at S_i. In addition, each S_i also maintain two arrays $lastsent_i$ and $lastrcvd_i$ of size n_s. The jth entry of $lastsent_i$, $lastsent_i[j]$, denotes the number of messages sent by S_i to S_j. Similarly, the jth entry of $lastrcvd_i$, $lastrcvd_i[j]$, denotes the number of messages sent by S_j that have been received at S_i.

Initially, all the entries in the matrices M_l, and arrays $lastsent_i$ and $lastrcvd_j$ are set to 0. To send a message m to another MH h_d, h_s first sends the message to its MSS S_i. Assume that h_d is in the cell of S_j. S_i increments $lastsent_i[j]$ by one and then sends $\langle m, M_s, lastsent_i[j]\rangle$ to S_j. After that S_i sets $M_s[i,j]$ to $lastsent_i[j]$.

S_j on receiving $\langle m, M, seqno\rangle$ from S_i meant for h_d first checks whether m is deliverable. m is *deliverable* if S_j has received all the messages on which m causally depends ($lastrcvd_j[k] \geq M[k,j]$ for all k), and there is no message destined for h_d on which m causally depends which is yet to be delivered to h_d (i.e. $\nexists\langle m', M', seqno'\rangle$ destined for h_d sent by S_k yet to be delivered such that $seqno' \leq M[k,j]$). If so, S_j transmits m to h_d. If m is not currently deliverable, it is kept in $rcvQ_j$, until it becomes deliverable. Like YHH, we do not update M_d immediately after delivering m to h_d, but we store m in $ackQ_d$. When h_d receives m, it sends back an acknowledge message, $ack(m)$, to S_j. On receiving $ack(m)$, S_j sets $M_d[i,j]$ to the maximum of its original value and $seqno$ (piggybacked on m). Then it sets each element in M_d to the maximum of its original value and the value of the corresponding element in M (also piggybacked on m). This prevents any outgoing message from h_d to become causally dependent on m that is sent before m is received by h_d. For more detailed description of the static module, please refer to [15]. Section 5 gives the formal condition implemented by our static module.

4.2 Handoff Module

In order to ensure causally ordered message delivery, some steps have to taken during handoff after an MH moves from one cell to another. This can be illustrated by the following example. Let m_1 and m_2, $m_1 \rightarrow_h m_2$, be both destined for the same MH, say h_1. Assume that h_1 moves from the cell of S_1 to the cell of S_2. Moreover, it leaves the cell before m_1 arrives at S_1. Also, assume that m_2 is sent to S_2. S_2 cannot decide based on M_1, the matrix for h_1, that there are messages in transition for h_1 sent to S_1. To ensure $\mathcal{CO}$, S_2 has to ascertain that all the messages for h_1 sent to S_1 have been delivered.

We now describe the handoff module. Each MH h_l maintains a *mobility number*, mbl_l, which is initially set to 0. It is incremented every time a mobile host moves. Intuitively, *mbl* denotes the number of times an MH has changed cell. In addition, every MSS maintains an array of 2-tuples, denoted by *cell*, with an entry for each MH. The lth entry of $cell_i$, $cell_i[l]$ is a 2-tuple $\langle mbl, mss\rangle$, where the value of $cell_i[l].mss$ represents S_i's knowledge of the location of h_l and the value of $cell_i[l].mbl$ indicates how "current" the knowledge is.

Consider a scenario when an MH h_l moves from the cell of S_i to the cell of S_j. After switching cell, h_l increments mbl_l and sends $register(mbl_l, S_i)$ message to S_j to inform S_j of its presence. Also, h_l retransmits the messages to S_j for which it did not receive the acknowledge message from its previous MSS S_i. On receiving this message h_l, S_j updates $cell_j[l]$ (its local knowledge about the location of h_l) and sends $handoff\_begin(h_l, mbl_l)$ message to S_i. The MSS S_i, on receiving $handoff\_begin(h_l, mbl_l)$ message, updates $cell_i[l]$ and sends $enable(h_l, M_l, ackQ_l)$ message to S_j. It then broadcasts $notify(h_l, mbl_l, S_j)$ message to all MSSs (except S_i and S_j), and waits for $last(h_l)$ message from all the MSSs to which it sent *notify* message. Meanwhile, if any message received by S_i for h_l becomes deliverable, S_i marks it as "old" and forwards it to S_j. $ackQ_l$ is the queue of messages that have been sent to h_l from S_i, but acknowledgment has not been received.

On receiving $enable(h_l, M_l, ackQ_l)$ message from S_i, S_j first delivers all the messages in $ackQ_l$. It also updates M_l assuming all the messages in $ackQ_l$ have been received at h_l. Then S_j starts sending the application messages on behalf of h_l. S_j also delivers all the messages for h_l that are marked "old" in the order in which the messages arrived. However, messages destined for h_l that are not marked "old" are queued in $rcvQ_j$.

An MSS S_k, on receiving $notify(h_l, mbl_l, S_l)$ message, updates $cell_k[l]$ and then sends $last(h_l)$ message to S_i. Observe that since the channels among all the MSSs are assumed to be FIFO, after S_i receives $last(h_l)$ message from S_k there are no messages in transition destined for h_l that are sent by S_k to S_i. On receiving $last(h_l)$ message from all the MSSs (to which *notify* message was sent), S_i sends $handoff\_over(h_l)$ message to S_j. The handoff terminates at S_j after S_j receives $handoff\_over(h_l)$ message. S_j can now start delivering messages to h_l. Meanwhile, if S_j receives $handoff\_begin(h_l)$ message from some other MSS before the current handoff terminates, S_j responds to the message only after the handoff terminates.

Since we do not assume that the messages in the wired network are causally ordered, it is possible that a message m destined for h_l is sent to S_i (the old MSS of h_l), whereas its causally preceding message m', also destined for h_l, is sent to S_j (the new MSS of h_l). In order to prevent this, an MSS piggybacks additional information on all the message that contain application messages: messages destined for an MH (may or may not be marked as "old") and *enable* messages. On these messages, an MSS piggybacks its local knowledge of the location of all the mobile hosts that have changed their cells since it last communicated with the other MSS. On receiving this information, the other MSS updates its knowledge of the location of the MHs (its *cell*) based on their mobility number. In the worst case, this extra overhead could be as large as $O(n_h)$. In practice, we expect it to be much smaller. Let t_{snd} denote the mean inter-message generation time and t_{mov} be the mean inter-switch time for an MH. Then, the average extra overhead for uniform communication pattern (every MH has equal probability of sending a message to every other MH) is $\approx O(\frac{t_{snd}}{t_{mov}} n_s^2)$.

Our handoff module is more efficient than the handoff module in $AV2$ and $AV3$ since we do not require the messages exchanged among the MSSs to be causally ordered. Formal description of the handoff module and its correctness proof can be found in [15].

5 Comparisons

In this section we first state the predicate that characterizes our static module. Then we present the conditions implemented by the static modules of $AV2$ and *YHH*. Finally, we provide a comparison between all the protocols.

5.1 Characterization of Static Module

The static module in Section 4.1 implements,

$$
\begin{aligned}
(\mathcal{CO}'')\ \ & \langle \exists\, m_k : \hat{m}_i.dst = \hat{m}_k.dst : \\
& \quad (\hat{m}_i.snd \preccurlyeq_s \hat{m}_k.snd) \wedge (m_k.snd \rightarrow_h m_j.snd)\rangle \ \Rightarrow \\
& \quad\quad \neg(m_j.dlv \prec_h m_i.dlv) \wedge \neg(\hat{m}_j.dlv \prec_s \hat{m}_i.rcv),
\end{aligned}
$$

where $e \preccurlyeq_s f$ iff $(e = f) \vee (e \prec_s f)$, under the assumption that the channels among MSSs are FIFO. Moreover, if the channels among MSSs are not FIFO then it implements,

$$\mathcal{CO}'' \wedge (\hat{m}_i.snd \prec_s \hat{m}_j.snd \Rightarrow \neg(\hat{m}_j.dlv \prec_s \hat{m}_i.rcv))$$

The formal proof is given in [15]. For convenience, let $\mathcal{FO}'' \overset{def}{=} \hat{m}_i.snd \prec_s \hat{m}_j.snd \Rightarrow \neg(\hat{m}_j.dlv \prec_s \hat{m}_i.rcv)$.

5.2 Discussion

The proposed static module implements $\mathcal{CO}'' \wedge \mathcal{FO}''$ which is weaker than $\mathcal{CO}'$ implemented by $AV2$ ($\mathcal{CO}' \Rightarrow \mathcal{CO}'' \wedge \mathcal{FO}''$). As a result, unnecessary delay in our protocol is lower than that imposed in $AV2$. In the worst case, our message overhead in the wired network is $O(n_s^2 + n_h)$ but we expect it to be closer to $O(n_s^2)$ in practice. Our storage overhead in each MSS is O$(k \times n_s^2)$, where k is the number of MHs currently in the cell of the MSS. Unlike $AV2$, our handoff protocol does not require causal ordering among application messages and messages sent as part of the handoff protocol. This further reduces the unnecessary delay compared to $AV2$.

PSR [10] is not suitable for systems where the number of mobile hosts dynamically changes because the structure of information carried by each message in their algorithm depends on the number of participating processes. In our protocol, the structure of the information carried by each message in the wired network does not vary with the number of MHs in the system. So, our protocol is more suitable for dynamic systems. PSR, however, incurs no unnecessary delay in message delivery.

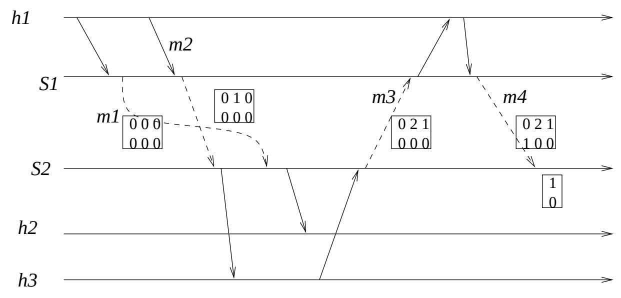

Figure 3: Liveness problem in *YHH*.

In Figure 3, we show a scenario where *YHH* does not satisfy liveness property. According to *YHH*, message m_4 will be delayed because $m_4.M[1,2] >$ MH_DELIV$_2[1]$. And since at the time when m_4 arrives at S_2, there are no messages in transit, m_4 is delayed indefinitely. The problem can be corrected by using sequence numbers. The static module in *YHH* (corrected) [16] satisfies $\hat{m}_i.snd \rightarrow_s \hat{m}_j.snd \Rightarrow \neg(m_j.dlv \prec_h m_i.dlv)$. Their message overhead in the wired network is $O(n_s \times n_h)$. This overhead is higher than ours but lower than $AV1$. Their unnecessary delay is strictly lower than $AV2$. When comparing in terms of unnecessary delay, their delay is lower than ours in the average case which is expected because of their higher message overhead. However, there are cases where our protocol does not impose delivery delay but their protocol does. One can further reduce the unnecessary delay in *YHH* using the technique introduced in this paper. By assigning a matrix of size $n_s \times n_h$ to each host, the condition implemented by their static module can be weakened to,

$$\begin{array}{l}\langle \exists\, m_k : m_i.dst = m_k.dst : \\ \quad (\hat{m}_i.snd \preccurlyeq_s \hat{m}_k.snd) \wedge (m_k.snd \rightarrow_h m_j.snd)\rangle \Rightarrow \\ \qquad \neg(m_j.dlv \prec_h m_i.dlv)\end{array}$$

Algorithm	Message overhead	Well-suited for dynamic systems
$AV2$	$O(n_s^2)$	Yes
PSR	$O(n_h^2)$	No
YHH	$O(n_s \times n_h)$	No
Our Algorithm	$O(n_s^2 + n_h)$	Yes

6 Performance Evaluation

Simulation experiments are conducted for different combinations of *message size* and *communication pattern.* We use 512 bytes for the size of small messages, and $8K - 10K$ bytes for large messages. Two communication patterns are used in the simulation: *uniform*, and *nonuniform.* Nonuniform pattern is induced by having odd numbered hosts generate messages at three times the rate of even numbered hosts. For each application message m, we define *MH-to-MH Delay* as the elapsed time between $m.snd$ and $m.dlv$. Similarly, *MSS-to-MSS Delay* is the elapsed time between $\hat{m}.snd$ and $\hat{m}.dlv$. The time between generation of successive messages at a mobile host is exponentially distributed with mean 100 ms. The throughput of a wired channel is assumed to be 100 Mbps, and the propagation delay is 7 ms. For a wireless channel, the throughput and propagation delay are respectively assumed to be

20 Mbps and 0.5 ms. This throughput is supported in European HiperLAN.

Results: Due to space limitation, we plot the MH-to-MH and MSS-to-MSS delay from our static module against those from $AV2$ in [15]. The simulation results show that our static module can reduce the MH-to-MH delay by as much as 18.4%, and MSS-to-MSS delay by 20.7% under uniform communication pattern and small message size. Under large message size, our static module can reduce the MH-to-MH delay by as much as 11.02%, and MSS-to-MSS delay by 18.7%.

Under nonuniform communication pattern and small message size, the result shows that our static module can reduce the MH-to-MH delay by as much as 18.9%, and MSS-to-MSS delay by 20.9%. For large message size, our static module can reduce the MH-to-MH delay by as much as 12.11%, and MSS-to-MSS delay by 19% .

7 Conclusion

We have presented a protocol that maintains the low message overhead while reducing unnecessary delivery delay imposed by $AV2$. Unlike PSR and YHH, our proposed protocol is scalable and suitable for dynamic systems. It is scalable because message overhead does not depend on the number of mobile hosts. And it is suitable for dynamic systems because it is easy to adapt to the changes in the number of participating mobile hosts. Delivery delay is reduced at the cost of higher storage space required on each MSS. Unlike $AV2$, our handoff protocol does not require causal ordering among application messages and messages sent as part of the handoff protocol. This further reduces the unnecessary delay in our protocol compared to $AV2$. In future, as the throughput of wireless links keeps increasing, the reduction of the end-to-end delay achieved by our protocol will also be higher.

References

[1] F. Adelstein and M. Singhal. Real-time Causal Message Ordering in Multimedia Systems. In *Proceedings of 15th International Conference on Distributed Computing Systems*, pages 36–43, June 1995.

[2] M. Ahamad, P. Hutto, and R. John. Implementing and Programming Causal Distributed Memory. In *Proceedings of the 11th IEEE International Conference on Distributed Computing Systems*, pages 271–281, 1991.

[3] S. Alagar and S. Venkatesan. Causal Ordering in Distributed Mobile Systems. *IEEE Transactions of Computers*, 6(3), March 1997.

[4] O. Babaoglu and K. Marzullo. Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanisms. In Sape Mullender, editor, *Distributed Systems*, pages 55–96. Addison-Wesley, 1993.

[5] K. Birman and T. Joseph. Reliable Communication in Presence of Failures. *ACM Transactions on Computer Systems*, 5(1):47–76, February 1987.

[6] K. Birman, A. Schiper, and P. Stepehenson. Lightweight causal and atomic broadcast. *ACM Transactions on Computer Systems*, 9(3):272–314, 1991.

[7] A.D. Kshemkalyani and M. Singhal. An Optimal Algorithm for Generalized Causal Message Ordering. In *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing*, pages 87–88, Philadelphia, Pennsylvania, May 1996.

[8] L. Lamport. Time, Clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[9] A. Mostefaoui and M. Raynal. Causal Multicasts in Overlapping Groups: Towards a Low Cost Approach. In *Proceedings of the 4th IEEE International Conference on Future Trends in Distributed Computing Systems*, pages 136–142, Lisbon, September 1993.

[10] R. Prakash, M. Raynal, and M. Singhal. An efficient causal ordering algorithm for mobile computing environments. In *Proceedings of the 16th International Conference on Distributed Computing Systems*, 1996.

[11] M. Raynal, A. Schiper, and S. Toueg. Causal Ordering abstraction and a simple way to implement it. In *Information Processing Letters*, volume 39(6), pages 343–350, 1991.

[12] M. Raynal, G. Thia-Kime, and M. Ahamad. From Serializable to Causal Transactions for Collaborative Applications. Technical Report 983, Irisa-Rennes, France, February 1996. 22 pages.

[13] L. Rodrigues and P. Verissimo. Causal Separators for Large-Scale Multicast Communication. In *Proceedings of the 15th IEEE International Conference on Distributed Computing Systems*, pages 83–91, Vancouver, June 1995.

[14] A. Schiper, J. Eggli, and A. Sandoz. A New Algorithm to Implement Causal Ordering. In *Proceedings of the 3rd International Workshop on Distributed Algorithms*, LNCS-392, pages 219–232, Berlin, 1989.

[15] C. Skawratananond, N. Mittal, and V. K. Garg. A Lightweight Algorithm for Causal Message Ordering in Mobile Computing Systems. Technical Report TR-PDS-1998-011, PDS Lab, University of Texas at Austin, USA, November 1998.

[16] Li-Hsing Yen, Ting-Lu Huang, and Shu-Yuen Hwang. A Protocol for Causally Ordered Message Delivery in Mobile Computing Systems. *Mobile Networks and Applications*, 1997.