Causal Multicast in Mobile Networks

Punit Chandra and Ajay D. Kshemkalyani
Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607, USA
{pchandra, ajayk}@cs.uic.edu

Abstract

Distributed applications using file sharing and file replication, conferencing, and delivery in multimedia systems often need to use the semantics of causal multicast. Causal multicast in a mobile system is complicated by the various limitations imposed by the mobile system. Though there are different directions to improve the performance of causal multicast algorithms in mobile systems, at the core of such algorithms is the module to impose causal delivery among a set of nodes – which may be either mobile hosts, or mobile service stations, or peers at a higher level in the interconnection hierarchy. We show how to adapt the optimal causal multicast algorithm of Kshemkalyani-Singhal to mobile networks, and then show simulation results comparing its performance with that of other algorithms for causal multicast in mobile networks.

1. Introduction

Semantic multicast is useful in areas such as managing replicated database updates, shared files, conferencing, collaborative tele-immersive sessions, and data delivery in multimedia systems. Let $Send(M)$ denote the event of a process handing over the message M to the communication subsystem. Let $Deliver(M)$ denote the event of M being *delivered* to a process after it is been received by its local communication subsystem. The system respects the semantics of *causal ordering* (CO) [5] iff for any messages M_1 and M_2 sent to the same destination, $(Send(M_1) \longrightarrow Send(M_2)) \Longrightarrow (Deliver(M_1) \longrightarrow Deliver(M_2))$. [1] Many causal ordering algorithms have been proposed. See [6, 8, 10, 12] for good surveys.

A causal ordering algorithm implementation has two forms of space overheads, viz., the size of control information on each message (message space overhead) and the size of buffer space at each process (log space overhead). The causal ordering algorithm by Raynal, Schiper and Toueg (RST) [16] is a canonical solution. It has a fixed message space overhead and log space overhead of n^2 integers, where n is the number of processes in the system. Kshemkalyani and Singhal identified and formulated the necessary and sufficient conditions on the information required for causal multicast, and gave an optimal algorithm (KS) to implement them [10, 11, 12]. This algorithm was proved to be optimal under all network conditions, but *without* making any simplifying system/communication assumptions [10, 11, 12]. Its message space, log space, and time overheads have the same pattern. Though the worst-case space complexity of the algorithm is $O(n^2)$ integers, extensive simulations under a wide range of conditions showed that the overhead is closer to being linear in n, and it has only a small dependence on (i) the network load, congestion, and message transmission times, (ii) frequency of communication, and (iii) percentage of multicasts [7].

Multicast in a mobile cellular network has received much recent attention. We assume a mobile cellular network consists of mobile hosts (MH) which communicate via static hosts called Mobile Support Stations (MSS) (see Fig. 1). For simplicity, it is assumed that the system consists of only MHs and MSSs. The MSSs are connected to each other by a high-speed wire-line network. The geographical coverage area under a MSS is called a *cell*. Each MH belongs to at most one cell at a time. The communication between a MH and its MSS is via a wireless channel. Whenever a MH moves from one cell to another, a hand-off procedure is executed between the two MSSs of the two cells involved. (For example, between MSSs of Cell A and Cell E in Fig. 1.) We denotes the number of MSSs and MHs by n_s and n_h, respectively. There are several algorithms for causal multicast in mobile cellular networks [2, 3, 14, 15, 18, 19].

Alagar and Venkatesan [2] presented a suite of algorithms that all use RST [16] to enforce causal order. These algorithms assume reliable wireless and wire-line networks. They use the fact that the wireless channel between MSS and each MH is FIFO, thus the log for each MH for causal ordering is maintained at the MSS level rather than at MHs.

[1] Events in a distributed execution are ordered by the causality relation [13], denoted by $\longrightarrow$. For two events $e1$ and $e2$, $e1 \longrightarrow e2$ iff (i) $e1$ and $e2$ occur on the same process and $e1$ occurs before $e2$, or (ii) $e1$ is the send of a message and $e2$ is the delivery of that message, or (iii) there exists an event $e3$ such that $e1 \longrightarrow e3$ and $e3 \longrightarrow e2$.

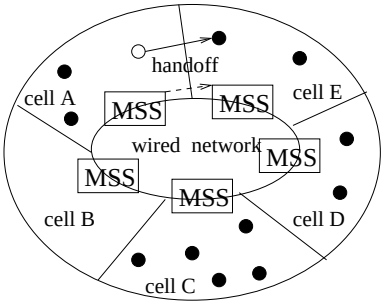

Figure 1. Mobile cellular network architecture.

AV-1 has a message overhead of $O(n_h^2)$ and requires $O(1)$ messages for hand-off. AV-2 has a low message space overhead of $O(n_s^2)$, and requires $O(n_s)$ messages (a broadcast) for hand-off. AV-2 is smart in realizing that it is sufficient only to maintain causal order among the MSSs; each MSS acts as proxy for all MHs in its cell. Note that in AV-2, a MSS does not have to broadcast a message to all the MSSs, but it does a broadcast to all MSSs only when a hand-off is executed. The algorithm in [18] is a variant of AV-2. The algorithm in [19] also adapts RST to achieve causal order in mobile networks. Its message space overhead is $O(n_s * n_h)$. The algorithm by Prakash, Raynal and Singhal (PRS) [15] combines an improved version of the RST causal order algorithm for static networks with that in [1], an algorithm for multicast with 'exactly-once delivery semantics' in mobile cellular networks. PRS assumes all wire-line and wireless networks to be reliable. Causal information is maintained at each MSS, for all the MHs in the system. The MSS broadcasts each message it receives from a MH in its cell, to all other MSSs. A MSS delivers the message to the destination MH if that MH is in its cell, once the message satisfies the causal order delivery condition. The message overhead of PRS is $O(n_h^2)$. An additional drawback is that MSSs have to broadcast each message.

The causal multicast algorithm presented in [3] uses coordinators at a level higher in a hierarchy than the MSSs. The coordinators broadcast the messages to all the MSSs which in turn broadcast them to all MHs. The decision of causal ordering delivery takes place at the MH. The wired network is assumed to be FIFO and reliable, while the wireless network is assumed to be FIFO but messages can be lost. The algorithm has a message space overhead of $O(m_c)$, where m_c is the number of coordinators in the system model. This algorithm is part of a larger suite of fault-tolerant message ordering algorithms in mobile systems, and hence the need for the coordinator nodes. The algorithm in [14] also does a broadcast among MSSs, and then within each cell to MHs. As broadcast is used in [3, 14], causal ordering requires $O(n)$ message space overhead on $O(n)$ messages constituting a broadcast, where n is the number of processors involved [4, 6]. Although the message complexity is linear in the number of MSS or coordinators, all the messages must be broadcast.

All the multicast algorithms for mobile systems discussed till now can be divided into two categories.

Category 1: Messages are sent using the point-to-point communication paradigm, as in [2, 15, 18, 19]. The causal order has to be maintained using a $O(n^2)$ message space overhead. Only one message needs to be sent to the MSS which has the destination MH and no broadcasts are needed except during hand-off, (although [15] always uses broadcasts).

Category 2: All the messages are broadcast, as in [3, 14]. Hence, an algorithm similar to [4, 6] can be used to achieve causal order. The message space overhead is linear in n, (per message). But broadcasting a message of size $O(n)$, unless implemented with a shared medium/hardware support, requires n point-to-point messages of size $O(n)$ each (or even $O(n^2)$ messages, depending on the implementation such as flooding). This results in at least a net $O(n^2)$ message space overhead per broadcast, in addition to the OS overhead of handling more messages.

Although these differ significantly, a *core module* for Category 1 is the *causal ordering module* for the static wire-line network connecting the MSSs. The focus of this work is to address the extent to which the $O(n^2)$ overhead – where n is n_s, n_h, or any equivalent usage – is reduced in practice for this network. We make the following contributions.

1. We show how to adapt the optimal KS algorithm to the *core causal ordering module* in mobile systems.
2. We then show via extensive simulations that KS greatly outperforms PRS which is a representative algorithm for mobile cellular networks.
3. From the above analysis, it is seen that the KS algorithm provides the best performance for the core CO module among Category 1 algorithms, and it competes closely with the (inflexible) Category 2 algorithms that broadcast each message even when not needed.

We stress that our focus is on the core CO module. KS can be used orthogonally with other directions, such as the following, to further improve performance.

- Use a hierarchical cluster organization [3, 4], where the CO module is used in each cluster.
- Send incremental changes, as in [17].
- Reduce falsely causally-ordered messages [9].

Section 2 outlines the RST algorithm, the KS algorithm, and the PRS algorithm. Section 3 adapts the KS algorithm to a mobile system. Section 4 gives the comparative simulation results. Section 5 concludes.

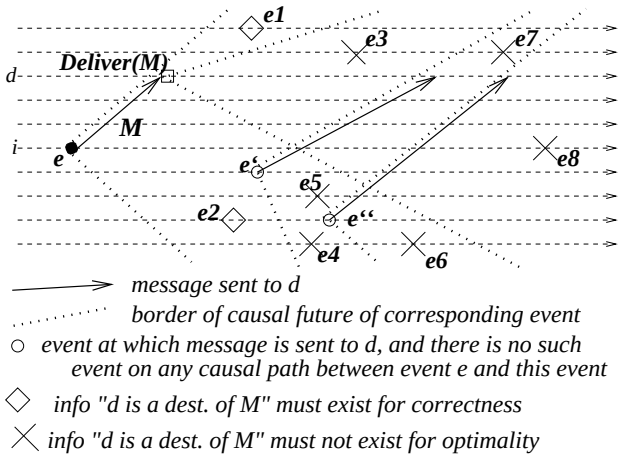

Figure 2. An example to illustrate the Propagation Constraints [10, 11, 12].

2. Overview of algorithms

2.1. The RST algorithm

Each of the n processes in the system maintains a $n \times n$ matrix, called the $SENT$ matrix. $SENT[i, j]$ is the process's best knowledge of the number of messages sent by process P_i to process P_j. A process also maintains an array $DELIV$ of size n, where $DELIV[k]$ is the number of messages sent by process P_k that have already been delivered locally. The $SENT$ matrix of the sender process is piggybacked on each message sent. When process P_j receives message M with the matrix SP piggybacked on it, the message is delivered only if, $\forall\, i$, $DELIV[i] \geq SP[i, j]$. P_j then updates its local matrix $SENT_j$ as: $\forall k \forall l \in \{1, \ldots, n\}$, $SENT_j[k, l] = max(SENT_j[k, l], SP[k, l])$. The space overhead on each message and in local storage at each process is the size of the matrix $SENT$, which is n^2 integers.

2.2. The KS algorithm

Kshemkalyani and Singhal identified the necessary and sufficient conditions on the information required for causal multicast, and proposed an algorithm (KS) to implement the conditions [10, 11, 12]. The causal past (resp., future) of an event e is the set $\{e' \mid e' \longrightarrow e\}$ (resp., $\{e' \mid e \longrightarrow e'\}$). A path in the computation graph is termed a *causal path*.

The KS algorithm achieves optimality by storing in local message logs and propagating on messages, information of the form "d is a destination of M" about a message M sent in the causal past, *as long as* and *only as long as*

(*Propagation Constraint I:*) it is not known that the message M is delivered to d, and

(*Propagation Constraint II:*) it is not known that a message has been sent to d in the causal future of $Send(M)$, and hence it is not guaranteed using a reasoning based on transitivity that the message M will be delivered to d in CO.

Let $M.Dests$ denote the set of destinations of M. The Propagation Constraints also imply that if either (I) or (II) is false, the information "$d \in M.Dests$" must *not* be stored or propagated, even to remember that (I) or (II) has been falsified. In addition to the Propagation Constraints, the algorithm follows a *Delivery Condition*: A message M^* that carries information "$d \in M.Dests$", where message M was sent to d in the causal past of $Send(M^*)$, is not delivered to d if M has not yet been delivered to d.

The Propagation Constraints are illustrated with the help of Fig. 2. The message M is sent by process i at event e to process d. The information "$d \in M.Dests$"

- must exist at $e1$ and $e2$ because (I) and (II) are true.
- must not exist at $e3$ because (I) is false
- must not exist at $e4, e5, e6$ because (II) is false
- must not exist at $e7, e8$ because (I) and (II) are false

$M.Dests$ can be represented in the local logs at processes and piggybacked on messages using the data structures shown in Fig. 3 [7]. Assuming that pid is an integer, the size of a `LogStruct` structure is $3 + size(dests)$ integers, where $size(X)$ is the number of elements in the set X. The log space overhead is the sum of the sizes of all the entries in the log. The size of overhead on a message is the size of the `MsgOvhdStruct` structure sent on it. This can be determined as $4 + size(dests) + SIZE(olog)$, where $SIZE(X)$ is the sum of the sizes of all the entries in the set X of `LogStruct`. The message and log space overheads of the KS algorithm, as measured using this data structure, are denoted by KS in our simulation.

Rather than storing the log as a variable length array of type `LogStruct`, a 2-dimensional bit array $logArray$ can be used to simplify the data structure. $logArray[i][j] = 1$ indicates the presence of an entry in the log, corresponding to the latest message sent from P_i to P_j. The modified log contains only the clock value of the send events of the messages whose entries are flagged in $logArray$. The total overhead is now $n^2/W +$ (number of entries in the log) integers, where W is the number of bits used to represent an integer. The message and log space overheads of the KS algorithm, based on this data structure and assuming $W = 32$, are denoted by KS'. The values for KS' depend on the value of n and on how sparse the log is.

2.3. Algorithms for cellular networks

With the emerging growth of mobile cellular networks, several recently proposed algorithms [2, 3, 14, 15, 18, 19] provided causal ordering in such networks. The algorithms [2, 15, 18, 19] use some variant of the RST algorithm, that

```
type LogStruct = record
    sender : pid;
    clock: integer;
    numdests: integer;
    dests: array[1..numdests] of pid;
end
```

```
type MsgOvhdStruct = record
    sender: pid;
    clock: integer;
    numdests: integer;
    numLogEntries: integer;
    dests: array[1..numdests] of pid;
    olog: array[1..numLogEntries] of LogStruct;
end
```

Figure 3. The log data structure and message overhead data structure [7] for KS.

is tailored for a mobile architecture. For example, the PRS algorithm [15] tracks direct predecessor messages using the mobile system framework of Acharya and Badrinath [1].

The multicast message overhead of PRS consists of (i) the vector CB, each entry of which is a set of up to n tuples of the form (pid, seq_no), (ii) counter $sent$, (iii) the list of destinations, and (iv) the pid of the sender. As each entry in CB is a set of tuples, the size of the vector CB is two times the number of tuples in CB. By using a n^2 bit string, the size of CB can be reduced to (number of tuples $+n^2/W$) integers, where W is the number of bits per integer. Thus,
$Total\_Msg\_Ovhd$
$= |i| + |sent_i| + |destination(M)| + |CB_i|$
$= 2 + |destination(M)| +$ (number of tuples $+ n^2/W$)

The log overhead at each processor is the matrix $Delivered$, vector CB, and counter $sent$. This adds to n^2 + (number of tuples $+n^2/W$) $+1$. W is assumed to be 32. The message overhead of PRS using the data structures to support the bit-string representation is denoted by PRS in the simulations.

3. Causal multicast for cellular networks

The KS causal multicast algorithm can be adapted to a mobile cellular network. Moving from a static to a mobile network presents several problems. The limitations of memory, computational power, and battery life of a mobile host (MH) require a minimum algorithm implementation at the MH. As the MH can move from one cell to another, it may not receive a message at all or it may receive multiple copies. Another issue is related with the lifetime of the messages at MSS, i.e., for how long should a message be cached. Acharya and Badrinath [1] present an algorithm that delivers a multicast message exactly once to a group of mobile destinations. PRS [15] presents a scheme that uses the above algorithm [1] coupled with its own causal ordering module to implement causal ordering for mobile networks. A similar approach can be used to combine KS and the above algorithm [1]. The added advantage of using KS is the significant decrease in message overhead for communication between MSSs.

3.1. KS algorithm adapted to cellular networks

1. Data structures: Every mobile host MH_h has a log log_h (see Section 2.2) and array $RECD_h[1..n_s]$ associated with it. These data structures are stored *only* at the (local) MSS in the cell. The log_h is used for maintaining causal order with respect to MH_h. The array $RECD_h[1..n_s]$, initialized to all zeros, is used to guarantee that each message is delivered once to MH_h. A MSS also maintains a sequence number M_{seq}, which is incremented each time after it forwards a message on behalf of any of the MHs in its cell.

Each message forwarded by a MSS MSS_{init} on behalf of any MH in its cell is associated with a message-id M_{id}, defined to be a tuple (MSS_{init}, M_{seq}).

2. Algorithm overview: All the processing and maintaining data structures for enforcing causal order, exactly once delivery, and hand-offs is done by the MSSs. Consider the sending of a message from MH_x to MH_y. MH_x sends the message to its MSS, MSS_{init}, which updates log_x for causal delivery, then appends M_{id} and log_x (Section 2.2) to the message, and broadcasts the message to all MSSs. Once the message is sent, MSS_{init} increments M_{seq} by one. When a MSS receives a message M destined for MH_y from another MSS, it tests the following conditions.

- Destination MH_y is present in the local cell
- $RECD_y[MSS_{init}] = M_{seq}$
- The Delivery Condition (Section 2.2) is satisfied for M

If these conditions are satisfied, the message is delivered to MH_y. For each MH_k that is not a destination, $RECD_k[MSS_{init}]$ is incremented. The message is also buffered, irrespective of the outcome of the tests. On receiving the message, MH_y sends an ack to its MSS, which now increments $RECD_y[MSS_{init}]$, updates log_y as per KS, deletes the buffered message, and forwards the ack to MSS_{init}. MSS_{init} then broadcasts message $Delete(M_{id})$ to all MSSs. When a MSS receives $Delete(M_{id})$, it clears any information about message M_{id} from its buffer.

Hand-off for MH_h is handled by passing $RECD_h[1..n_s]$ and log_h from the old MSS to the new MSS. The transfer is delayed if a message has been delivered to MH_h but the update of $RECD_h[1..n_s]$ and log_h have not taken place.

3.2. Other adaptations of KS to cellular networks

The above adaptation was based on [1] to show an adaptation similar to PRS. KS can be adapted to cellular networks in more efficient ways.

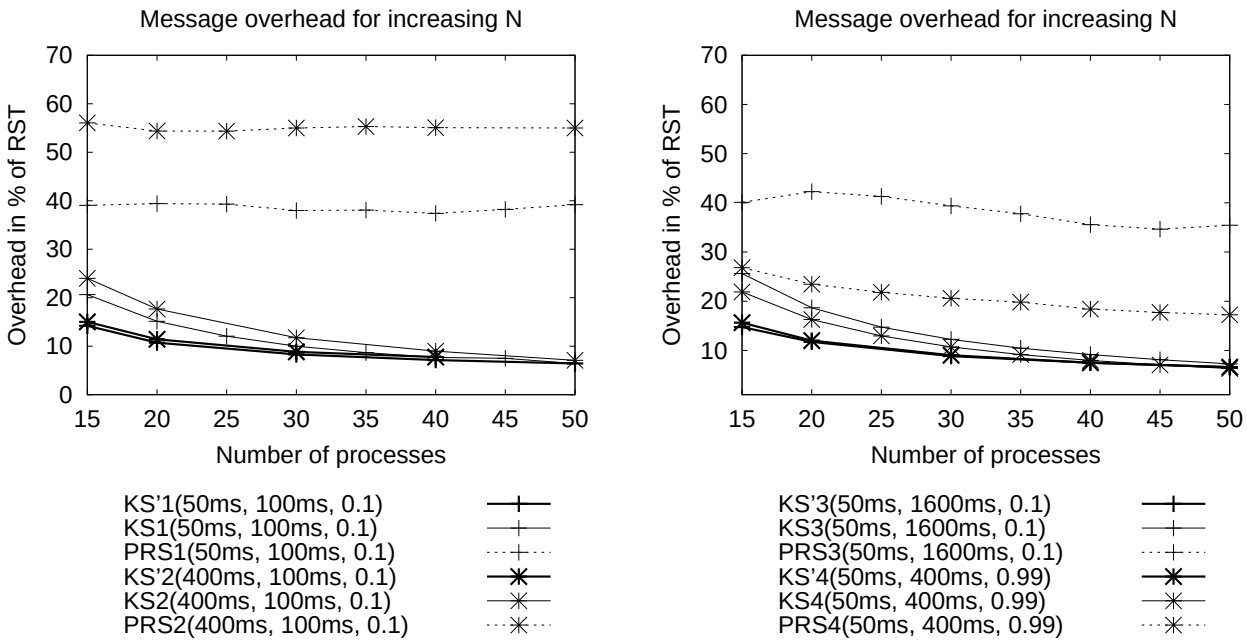

Figure 4. Average message overhead as a function of n.

- Causal message delivery can be implemented by sending messages directly between MHs through wireless (even ad-hoc) networks. In this case, the local logs can be maintained by the MHs themselves. The low bandwidth of wireless networks and the low memory capacity of MHs makes KS most suited for the core module for causal multicast.
- KS can be used as the core CO module with other architectures for mobile networks, such as AV-2 [2]. Instead of the RST algorithm, the use of KS will lead to substantial decrease in the message overhead. There is no need to broadcast each message to all MSSs. The message is only sent to the MSS containing the recipient MH, using a location management scheme. Broadcast is used only during hand-off.

4. Simulation results

In the simulations of KS, KS', and PRS, we assumed w.l.o.g. that each MH runs a single process and that a reliable communication network delivers messages in FIFO order for any pair of processes (MHs and MSSs). The simulation used Java2 1.4.1 SDK with Object Space JGL 4.0. The simulation experiment setup and methodology are similar to [7]; hence details are omitted.

The following system parameters are of interest.

- **Number of processes (n)**.
- **Mean inter-message time (*MIMT*)**: The inter-message time is the average time between two message send events. It is exponentially distributed about the *MIMT*.
- **Mean transmission time (*MTT*):** The transmission time of a message here implicitly refers to the *msg. size/bandwidth + propagation delay*. We model this time as an exponential distribution about the mean, *MTT*. The *MTT* captures the various network distances, available bandwidth, and congestion levels. For the purpose of enforcing this mean, multicasts are treated as multiple unicasts and transmission time is independently determined for each unicast.
- **Multicast frequency (*M/T*):** The ratio of the number of multicasts to the total number of message sends *(M/T)* is the parameter on the basis of which the multicast sensitivity of the algorithms can be determined. The number of destinations of a multicast is given by a uniform distribution ranging from 1 to n.

The performance metrics used are the following.

- The average number of integers sent per message under various combinations of the system parameters, viz., n, *MTT*, *MIMT*, and *M/T*.
- The average size of the log in integers.

Simulation experiments were conducted for different combinations of the parameters. For each combination, ten runs were executed; the results of the ten runs did not differ from each other by more than a percent. Hence, only the mean is reported for each combination and the variance is not reported. All the overheads are reported as a percentage of the corresponding deterministic overhead n^2 of the RST algorithm.

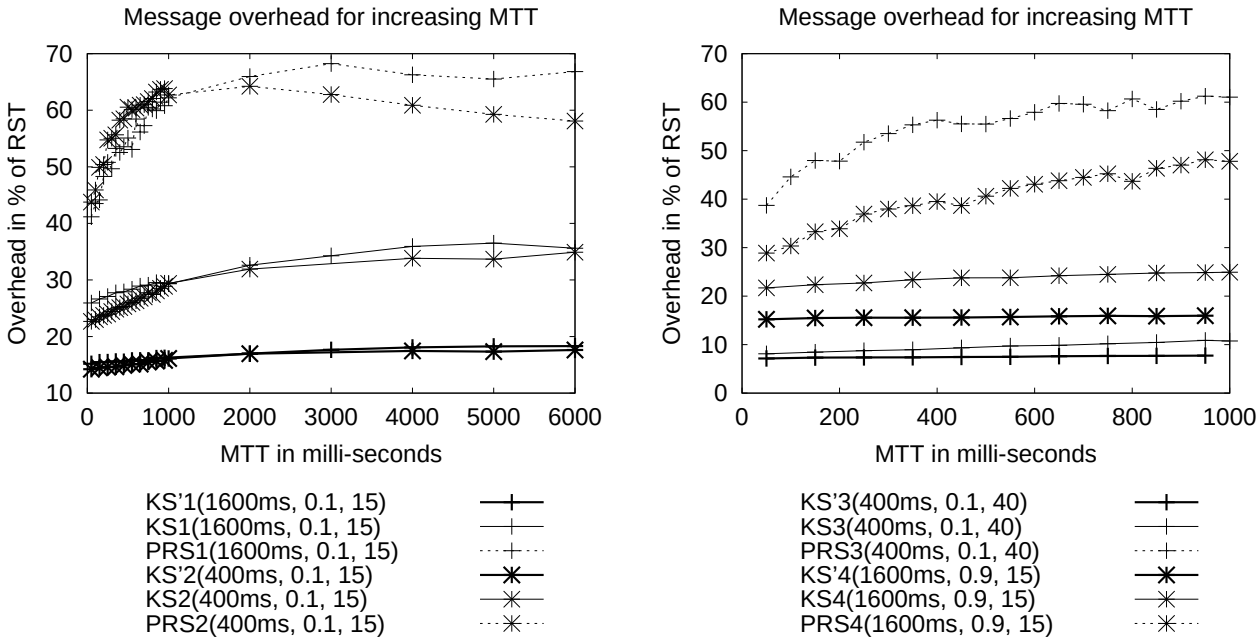

Figure 5. Average message overhead as a function of *MTT*.

It was seen that for KS and KS', the log space overhead and time overhead followed the same pattern as the message space overhead in all the experiments. For PRS, the log space overhead is much worse than even the n^2 required by RST. So we focus on the message space overhead comparisons, and the log space overhead plots are not shown. See [7] for a detailed analysis of the behavior of KS and KS'.

4.1. Scalability with increasing n

The simulations were performed for KS, KS', and PRS for the combinations of the parameters (*MTT, MIMT, M/T*) fixed at $(50ms, 100ms, 0.1)$, $(400ms, 100ms, 0.1)$, $(50ms, 1600ms, 0.1)$, and $(50ms, 400ms, 0.99)$ while n was varied up to 50 processors. For larger systems, a hierarchical organization is typically used, where the number of processes n in each cluster can be controlled. These four combinations of the parameter settings correspond to a *MIMT* to *MTT* ratio of 2, 0.25, 32, and 8, resp.

The average message overhead is shown in Fig. 4. KS and KS' perform significantly better than PRS under all the network conditions simulated. The curves for KS and KS' show a similar trend, and are very close together for high values of n. KS and KS' are almost linear in n. As expected for lower values of n, KS' is more efficient than KS. At higher values of n, KS' approaches KS because the log is very sparse and the n^2 bit-array adds more overhead than it reduces. We expect that for any set of traffic parameters, eventually as n keeps increasing, some threshold value will be reached beyond which KS performs better than KS'.

4.2. Impact of increasing transmission time

Increasing *MTT* is indicative of larger distances, decrease in available bandwidth, and increasing network congestion. The simulations were performed for KS, KS', and PRS with the parameters (*MIMT, M/T*, n) fixed at $(1600ms, 0.1, 15)$, $(400ms, 0.1, 15)$, $(400ms, 0.1, 40)$, and $(1600ms, 0.9, 15)$, respectively. The *MTT* was increased from $50ms$ to $6000ms$ for the first two cases, while it has a range of $50ms$ to $1000ms$ for the later two cases. Thus, the four cases had a *MIMT* to *MMT* ratio varied from 32 to 0.3, 8 to 0.067, 8 to 0.4, and 32 to 1.6, respectively. The average message overhead is shown in Fig. 5.

KS and KS' outperform PRS significantly in all cases, and KS' shows a lower overhead than KS. KS and KS' are not much sensitive to the *MTT* parameter.

4.3. Impact of decreasing communication load

This set of simulations studies the overhead behavior of KS, KS', PRS as a function of usage of communication. The values of (*MTT, M/T*, n) were fixed at $(50ms, 0.1, 40)$, $(50ms, 0.1, 20)$, $(400ms, 0.1, 20)$, and $(50ms, 0.9, 20)$. *MIMT* is varied from $50ms$ to $1200ms$ for the first two cases and from $50ms$ to $12000ms$ for the other two. Thus, the four cases had a *MIMT* to *MMT* ratio varied from 1 to 24, 1 to 24, 0.125 to 30, and 1 to 240, respectively. The average message overhead is shown in Fig. 6. Overall, KS and KS' perform much better than PRS. KS and KS' are not much sensitive to the *MIMT* parameter.

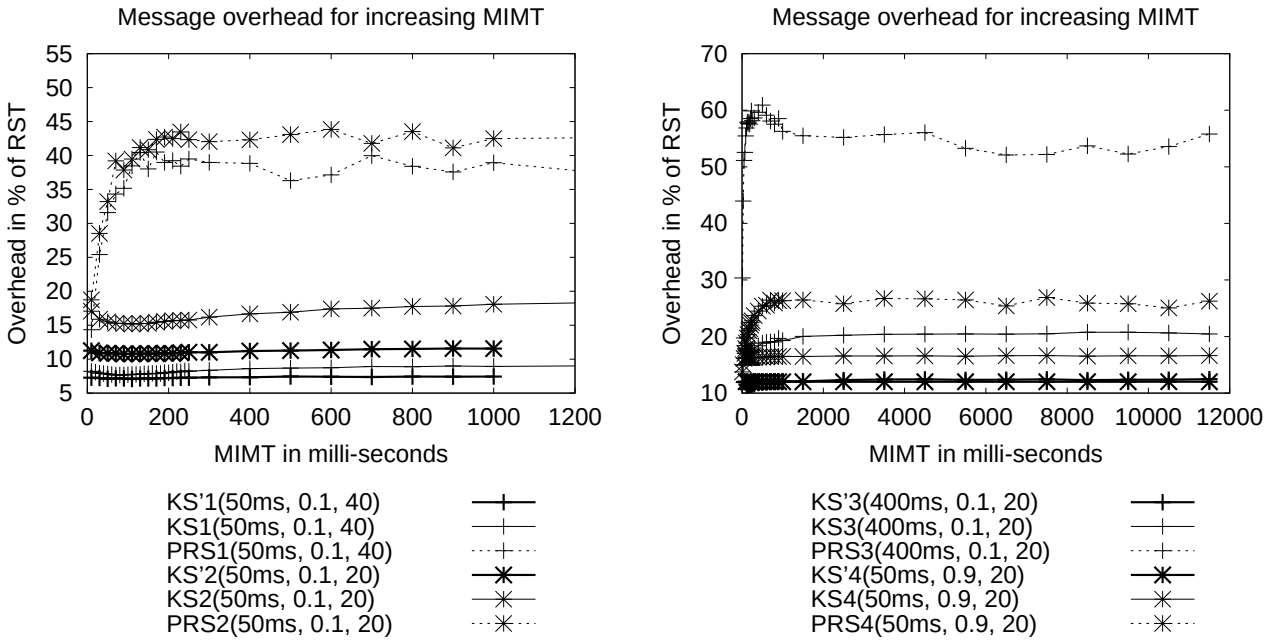

Figure 6. Average message overhead as a function of *MIMT*.

4.4. Impact of increasing multicast frequency

This experiment studies the sensitivity of KS, KS', and PRS to multicast frequency. We ran simulations for KS, KS', and PRS, increasing *M/T* from 0.1 to 1.0 in steps of 0.1. The values of (*MTT*, *MIMT*, n) were fixed at $(75ms, 500ms, 12)$, $(50ms, 500ms, 40)$, $(50ms, 400ms, 20)$, and $(50ms, 1600, 15)$. *MTT* was increased from $50ms$ to $75ms$ across the four runs. *MIMT* was varied from $500ms$ to $1600ms$. Thus, the four cases had a *MIMT* to *MMT* ratio of 6.67, 10, 8, and 32, respectively. The average message overhead is shown in Fig. 7.

KS and KS' are not much sensitive to the *M/T* parameter. The decrease in the overhead is much more for PRS than for KS or KS' – PRS gradually begins to approach KS as multicast frequency increases. In the extreme case of full broadcasts, the behavior will tend to be closer. Even though there is a decline in the overhead for PRS with increasing *M/T*, KS and KS' are still significantly more efficient than PRS. This is even more evident for a larger number of processes.

5. Concluding discussion

Efficient causal multicast support is important for applications such as multiplayer gaming, text conferencing, distributed multimedia interactions, and event notifications. We first showed how to adapt the KS algorithm [10, 11, 12], which has been proved to be optimal, to mobile cellular networks. The adapted KS algorithm can be used for the core causal ordering module. We then performed extensive simulations to compare the overhead incurred by two implementations of the KS algorithm – KS and KS' – and the PRS algorithm, for mobile cellular networks. KS and KS' were seen to have significantly lower overhead in all respects. Specifically, the low message space overhead of the KS algorithm is seen to be

- fairly independent of *MIMT*, *MTT*, and *M/T*.
- somewhat linear in n, making the KS algorithm very scalable.

It can thus be concluded that for causal multicasts:

- The KS algorithm provides the best performance for the core module used by the class of algorithms ([2, 15, 18, 19]) that send a point-to-point message between each pair of MSSs involved.
- The KS algorithm competes closely with the class of algorithms ([3, 14]) that do a broadcast, (which requires $O(n)$ messages of size $O(n)$ each per broadcast), even when a broadcast is not required.
- The KS algorithm can be used in addition to other techniques (such as hierarchical clustering, transmitting incremental updates, and reducing false causal relationships) to improve CO multicast performance.

Acknowledgements: This work was supported by the U.S. National Science Foundation grant CCR-9875617.

References

[1] A. Acharya, B. Badrinath, A framework for delivering multicast messages in networks with mobile hosts, *ACM/Baltzer Mobile Networks and Applications*, 1(2): 199-219, June 1996.

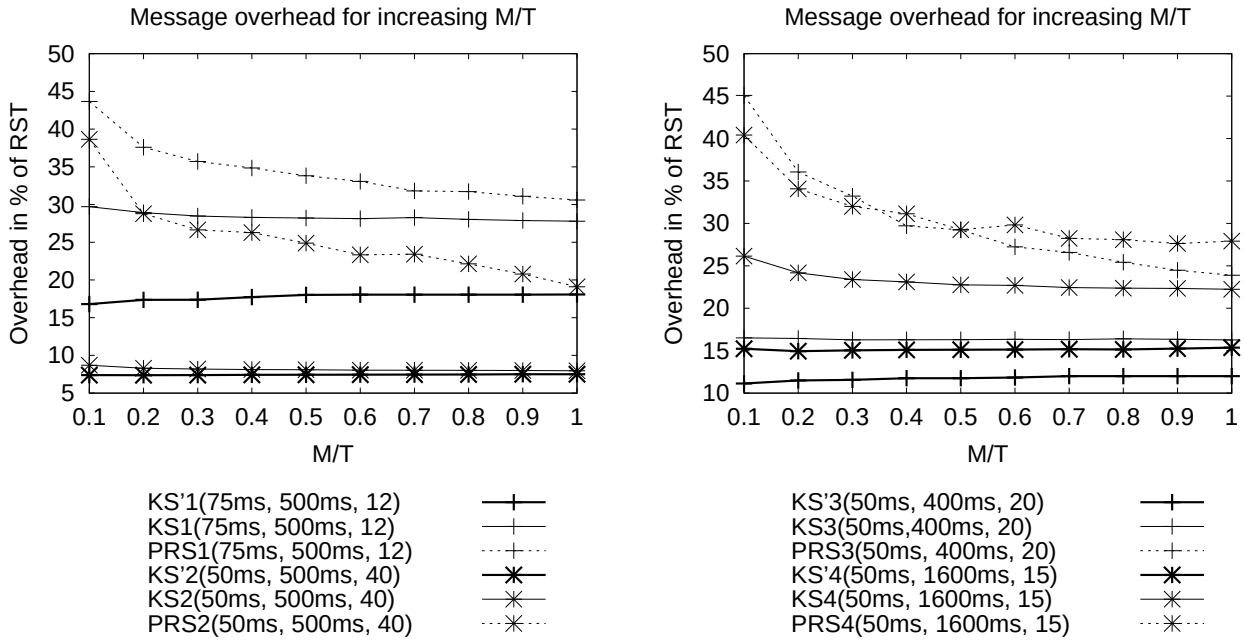

Figure 7. Average message overhead as a function of *M/T*.

[2] S. Alagar, S. Venkatesan, Causal ordering in distributed mobile systems, *IEEE Trans. Computers,* 46(3): 353-361, Mar. 1997.

[3] G. Anastasi, A. Bartoli, F. Spadoni, A reliable multicast protocol for distributed mobile systems: Design and evaluation, *IEEE Trans. Parallel and Distributed Systems,* 12(10): 1009-1022, Oct. 2001.

[4] R. Baldoni, R. Beraldi, R. Friedman, R. van Renesse, The hierarchical daisy architecture for causal delivery, *IEE/IOP Distributed Systems Engineering,* 6: 71-81, 1999.

[5] K. Birman, T. Joseph, Reliable communication in the presence of failures, *ACM Trans. Computer Systems*, 5(1): 47-76, Feb. 1987.

[6] K. Birman, A. Schiper, P. Stephenson, Lightweight causal and atomic group multicast, *ACM Trans. Computer Systems*, 9(3): 272-314, Aug. 1991.

[7] P. Chandra, P. Gambhire, A. D. Kshemkalyani, Performance of the optimal causal multicast algorithm: A statistical analysis, *IEEE Trans. on Parallel and Distributed Systems*, 14(1): 40-52, January 2004.

[8] G. Chockler, I. Keidar, R. Vitenberg, Group communication specifications: A comprehensive study, *ACM Computing Surveys,* 33(4): 1-43, Dec. 2001.

[9] P. Gambhire, A.D. Kshemkalyani, Reducing false causality in causal message ordering, *7th International High Performance Computing Conference*, LNCS 1970, Springer-Verlag, 83-95, December 2000.

[10] A.D. Kshemkalyani, M. Singhal, Necessary and sufficient conditions on information for causal message ordering and their optimal implementation, *Technical Report 29.2040*, IBM Research Triangle Park, July 1995. (Also Technical Report CISRC-7/95/TR33, July 1995, Department of Computer Science, Ohio State University.)

[11] A.D. Kshemkalyani, M. Singhal, An optimal algorithm for generalized causal message ordering, *15th ACM Symposium on Principles of Distributed Computing*, 87, May 1996.

[12] A.D. Kshemkalyani, M. Singhal, Necessary and sufficient conditions on information for causal message ordering and their optimal implementation, *Distributed Computing*, 11(2): 91-111, April 1998.

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

[14] C. Li, T. Huang, A mobile-support-station-based causal multicast algorithm in mobile computing environment, *Proc. Nat. Science Council, ROC(A),* 23(1): 100-110, 1999.

[15] R. Prakash, M. Raynal, M. Singhal, An adaptive causal ordering algorithm suited to mobile computing environments, *J. Parallel and Distributed Computing,* 41(2): 190-204, 1997.

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

[17] M. Singhal, A.D. Kshemkalyani, Efficient implementation of vector clocks, *Information Processing Letters*, 43(1): 47-52, August 1992.

[18] C. Skawratananond, N. Mittal, V. Garg, A lightweight algorithm for causal message ordering in mobile computing systems, *12th ISCA Conf. on Parallel and Distributed Computing Systems*, 245-250, 1999.

[19] L. Yen, T. Huang, S. Hwang, A protocol for causally ordered message delivery in mobile computing systems, *ACM/Baltzer Mobile Networks and Applications,* 2(4): 365-372, 1997.