

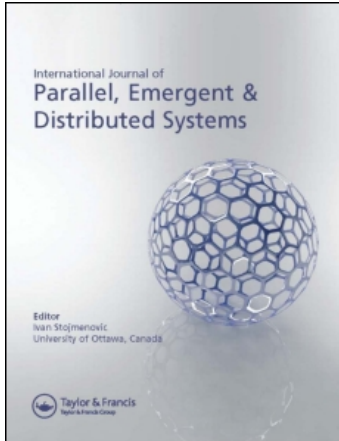
This article was downloaded by: [Mohamed, Maluk]

On: 20 May 2010

Access details: Access Details: [subscription number 921473019]

Publisher Taylor & Francis

Informa Ltd Registered in England and Wales Registered Number: 1072954 Registered office: Mortimer House, 37-41 Mortimer Street, London W1T 3JH, UK



International Journal of Parallel, Emergent and Distributed Systems

Publication details, including instructions for authors and subscription information:

<http://www.informaworld.com/smpp/title~content=t713729127>

EOMP: an exactly once multicast protocol for distributed mobile systems

M.A. Maluk Mohamed ^a; V. R. Devanathan ^a; D. Janakiram ^a

^a Distributed and Object Systems Laboratory, Department of Computer Science and Engineering, Indian Institute of Technology, Madras, India

Online publication date: 20 April 2010

To cite this Article Mohamed, M.A. Maluk , Devanathan, V. R. and Janakiram, D.(2010) 'EOMP: an exactly once multicast protocol for distributed mobile systems', International Journal of Parallel, Emergent and Distributed Systems, 25: 3, 183 – 207

To link to this Article: DOI: 10.1080/17445760903284665

URL: <http://dx.doi.org/10.1080/17445760903284665>

PLEASE SCROLL DOWN FOR ARTICLE

Full terms and conditions of use: <http://www.informaworld.com/terms-and-conditions-of-access.pdf>

This article may be used for research, teaching and private study purposes. Any substantial or systematic reproduction, re-distribution, re-selling, loan or sub-licensing, systematic supply or distribution in any form to anyone is expressly forbidden.

The publisher does not give any warranty express or implied or make any representation that the contents will be complete or accurate or up to date. The accuracy of any instructions, formulae and drug doses should be independently verified with primary sources. The publisher shall not be liable for any loss, actions, claims, proceedings, demand or costs or damages whatsoever or howsoever caused arising directly or indirectly in connection with or arising out of the use of this material.

EOMP: an exactly once multicast protocol for distributed mobile systems

M.A. Maluk Mohamed*, V.R. Devanathan¹ and D. Janakiram²

*Distributed and Object Systems Laboratory, Department of Computer Science and Engineering,
Indian Institute of Technology, Madras, India*

(Received 26 June 2008; final version received 20 August 2009)

Reliable multicast provides an elegant mechanism for structuring communication among closely cooperating entities. In a distributed mobile computing environment, disconnections and physical mobility of the mobile hosts (MHs) may result in the loss of messages. This makes the problem of ensuring reliable multicast difficult for MHs. Apart from ensuring reliable message delivery, guaranteeing exactly once message delivery greatly helps in conserving the battery power. Moreover, for transaction-based applications, total ordering of messages are semantically essential. In this report, an exactly once multicast protocol (called EOMP) with totally ordered message delivery is proposed. An unreliable wireless MAC level multicast is used for delivering the messages to the MHs and a novel mechanism for multicasting messages in the cellular network is proposed. An EOMP tolerates mobile support station (MSS) failure by making the MSS stateless. Dynamic host group membership is also supported by an EOMP. The performance of EOMP has been evaluated and compared with an existing an EOMP by simulation. The results of the simulation show that an EOMP has lower delay, lesser handoff, reduced buffer space overhead, and are more scalable. To establish the effectiveness of a reliable multicast protocol with totally ordered message delivery for mobile systems, two different applications viz., a real-time distributed collaborative text editor and a distributed image rendering application, were developed as case studies. The applications were easy to develop as mobility and disconnection were hidden from the application by an EOMP. The applications are also efficient as they use a powerful reliable multicast primitive for communication.

Keywords: mobile communication; reliable multicast; cellular multicasting; totally ordered message delivery; exactly once message delivery

1. Introduction

Mobile devices have made a significant impact in personal computing. Typically, mobile hosts (MHs) communicate with mobile support station (MSS) over a wireless medium and MSS serves as an access point for the MH. MHs are characterised by severe constraints such as limited battery power, limited bandwidth, frequent disconnections and physical mobility from one cell³ to another. Cells may not provide complete spatial coverage. MHs may move unpredictably and may be out-of-range, for a possibly long time, before entering another cell. These constraints may cause MH to lose messages even in an otherwise reliable error-free environment.

*Corresponding author. Email: maluk@cs.iitm.ernet.in

The recent popularity of many mobile devices, has increased the wide use of networks with MHs for group communication applications. To name few such applications are transmission of stock market quotes, transmission of television programmes, transmission of weather advisories, etc. To provide such applications, a protocol must be able to provide efficient multicast, guarantee delivery of all messages and allow users to belong to several multicast group, simultaneously. Multicast is a point-to-multipoint communication mechanism by which a message sent to the multicast group reaches all the group members. Distributed computing environment involving closely cooperating entities, reliable multicast mechanism provides an efficient way of structuring point-to-multipoint communication among the entities. Providing reliable multicast for mobile systems is more difficult due to the severe constraints and the erroneous nature of the wireless medium. In addition to ensuring reliability, guaranteeing exactly once message delivery to the MH is essential to conserve its battery power. Moreover, in many distributed applications that are based on transactions, totally ordered message delivery is semantically essential. Total ordering of messages means that for any two messages m_1 and m_2 , every MH either receives m_1 first and then m_2 or vice-versa.

In this paper, we propose an exactly once multicast protocol (EOMP), a protocol with totally ordered message delivery. Our protocol scenario includes cell phones and other such non-IP devices in addition to IP-based devices such as laptops. Hence, our protocol does not rely upon MobileIP [17] for routing the messages to MH. Our protocol uses a wireless MAC multicast to deliver multicast messages to MH. As cell phones are also a part of our system, and multicast for cellular networks is not available, as far as we know. Hence, we propose a MAC multicast model for cellular networks.

The salient features of our protocol are:

- A totally ordered EOMP with dynamic join/leave of groups by MHs.
- A *Novel* model for multicasting the messages in a wireless medium for cellular networks.
- Tolerates MSS failures.
- Minimal handoff overhead upon cell switching.
- Size of message headers, size of data structures at MH and MSS, and number of messages in the wired network are independent of the total number of MHs in the system.
- Scalability.

As achieving an exactly once message delivery is the prime motive of our protocol (other motive being total ordering), we compare by simulation, the performance of our protocol with RelM protocol of [6]. The rest of the paper is organised as follows: Section 2 discusses the related work in brief. Section 3 gives an overview of the system. In Section 4, the multicast model for cellular networks is explained. In Section 5, our protocol is described. In Section 6, the simulation model and the results of the simulation are discussed. Section 7 discusses a distributed collaborative application developed over an EOMP as the first case study. Section 8 presents a distributed image rendering application developed as the second case study, over an EOMP and Section 9 concludes the paper.

2. Related work

Refs. [1,4,6] are some existing reliable multicast protocols for mobile systems which guarantee an exactly once message delivery.

A two-tier approach consisting of MSS and MH is discussed in protocol [1], and it supports only static host (SH) group membership and FIFO ordering of messages. Furthermore, this protocol assumes a failure-free MSS model.

Another protocol supporting static group membership and FIFO ordering of messages is presented in [4]. It works on top of a location directory service, which indicates the current MSS associated with a given MH. Also, tracking of each individual MH is needed, which is very costly. Our protocol does not need any location directory service and tracking of individual MH is not done.

Protocol [6] is ReIM which we simulated for comparison. ReIM also uses a three-tier approach to ensure exactly once. Though ReIM has lesser handoff overhead than protocol [1], still considerable amount of messages are transferred among the SHs upon cell switching. ReIM supports dynamic join/leave of groups by MHs and seems to assume a complete spatial coverage of cells. Moreover, only SHs are the source of the messages.

However, none of these protocols provide totally ordered message delivery and all of them use an extensive handoff mechanism by forwarding the messages as the MH switches cells. Further, all these protocols use an unicast to deliver messages to the MH. Moreover, the number of messages in the wired network increases with the number of mobile receivers and hence, they do not scale well. As opposed to this, our protocol has a minimal handoff, uses wireless multicast and is scalable. Further, compared to these EOMPs, our protocol supports total ordering without much overhead and even tolerates MSS failures. Our protocol supports dynamic host group membership.

Protocol [2] uses broadcasting in the wireless medium to transfer messages from the MSS to the MH and does not provide the exactly once guarantee. It provides three increasingly strong guarantees viz. FIFO, causal and total ordering of messages. Though no handoff is used, the data structures at MSS depend on the number of MHs in the system. This protocol was simulated only for a single group, hence the number of unwanted messages was only due to duplicates. However, in a typical scenario consisting of multiple groups, the number of unwanted messages will increase with the number of groups in the system. Our protocol uses multicasting in a wireless medium and guarantees an exactly once message delivery. Data structures at MSS are independent of the number of MHs in the system.

Reliable MAC level multicast in wireless LANs are dealt in [10,22]. Protocol [10] uses a leader-based multicast, where each LAN has a leader for every group and multicast messages are sent to the leader and all the receivers passively listen to these messages. In case of message loss at the receiver, it sends a NACK that collides with ACK of the leader, thereby causing the sender to retransmit. Protocol [22] is specific to 802.11 LAN, where the contention phase for the multicast is reduced to one. Having a coordination mechanism in the transmission of ACK and Clear To Send messages, avoids collisions. These protocols are very specific to wireless LANs and operate at MAC layer.

A multicast protocol exploiting packet loss locality through caching is presented in [13]. This protocol is designed on the notion that packet losses are bursty and hence caching can aid in recovery of losses. In RingNet [24], a combination of logical trees and logical rings is used for multicast communication in mobile internet. A scalable method to improve packet delivery of multicast routing protocols in *ad hoc* networks is addressed in [7]. As a single forwarding path is vulnerable to node failures in a mobile *ad hoc* network, a ODMRP-based wireless multicast protocol named RODMRP is suggested in [16]. This protocol offers more reliable forwarding paths in face of node and network failures. However, these protocols do not support an exactly once delivery of messages.

Mobile multicast support using an old foreign agent [19] with the assistance of a MH's old foreign agent, routes the missing datagrams due to handoff in the adjacent network via tunnelling. Range-based mobile multicast [12] intends to trade off between the shortest delivery path and the frequency of the multicast tree reconfiguration by controlling the service range of the multicast home agent. In [11], a modified join-and-leave mechanism is employed and the missing data sequence is routed to other agents in the adjacent networks via tunnelling. Despite many proposals, the impact of tunnelling on multicast efficiency has not yet revealed sufficiently [9].

The protocol in [15] proposes how to achieve reliable an exactly once multicast communication considering the system as a two-tier model and it also provides fault tolerant support. However, it does not give the experimental results and the details related to correctness of the protocol. Our protocol is a general transport-level protocol. Our protocol is general in the sense that, it is not restricted to any specific mobile systems or IP-based mobile networks, and can be implemented even for cellular and other non-IP mobile networks.

3. System model

Our protocol uses a three layered model consisting of MHs at the lowermost layer, MSSs at the middle layer and coordinators and other SHs at the topmost layer and is shown in Figure 1. A MH communicates with its MSS only by message passing over the wireless interface. A MSS may broadcast messages to all MHs in its cell, multicast to those subscribed for a specific group or unicast messages to a MH. A MSS may also send messages over the wired network to a SH designated as coordinator. A coordinator is assigned the responsibility of a particular region and can communicate with MSSs under its region or other SHs via a wired network interface.

The assumptions about our model are:

- (1) Communication over the wired network is reliable and FIFO ordered.
- (2) Coordinators do not fail. Coordinators can be made to tolerate failures by primary-backup, hot-standby or by any other fault-tolerant mechanisms.

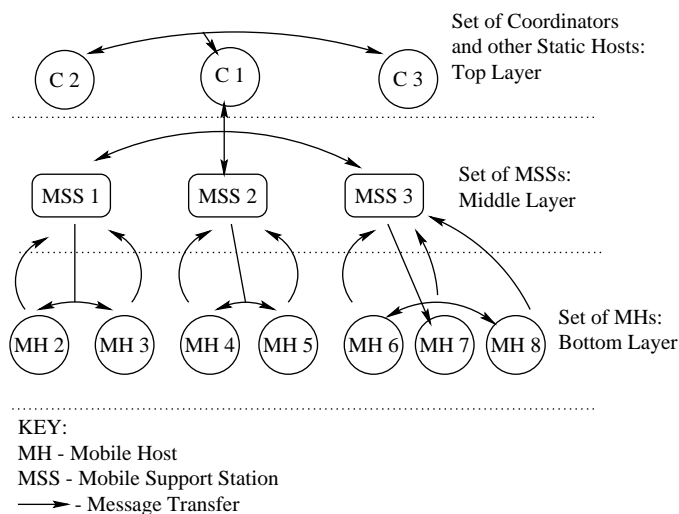


Figure 1. System model.

- (3) A MSS may fail and it does so by crashing, which is fail-stop (i.e. MSS does not behave maliciously). Hence, when a MSS fails, all MHs under its cell become out-of-range and either all MHs eventually enter another cell or the failed MSS is eventually replaced.
- (4) MHs do not fail. As the MHs are constrained devices, the data structures associated with the MHs are maintained in the MSS. Thus, even when the mobile devices go down due to battery failure, data are not lost.
- (5) MHs do not remain out-of-range forever and they eventually enter a cell after being out-of-range for possibly a long period.

These assumptions are realistic and are not difficult to achieve. For example, FIFO order in wired network can be achieved by using protocols such as TCP or by using a sliding window protocol [23].

4. An unreliable wireless multicast model for cellular networks

Traditionally, cellular networks provide only unicast and broadcast services. In this section, we propose a $1 \times N$ multicast model for multicasting the messages over cellular networks. First, briefly we will discuss the existing channel establishment and release (or teardown) procedures in the traditional unicast model for cellular networks and then propose our unreliable multicast model. A detailed discussion of unicast channel establishment and release can be found in [8,20].

4.1 Unicast communication

For unicast communication in cellular networks, the calls may be of two different types viz. mobile destination call and mobile origination call. The channel assignment for both these calls is similar with only few minor differences.

For channel assignment in mobile destination call, a MSS initiates with a paging request for the MH and MH replies by sending a channel request message. Then, the MH is authenticated and a channel⁴ is assigned for communication. In the case of mobile originating call, the procedure is similar except that the MH initiates with a channel request message.

An unicast communication guarantees reliable connection-oriented service, it is very important to avoid wrongly disconnecting an ongoing communication. Hence, the channel release involves exchange of two sets of confirming acknowledgements before releasing the channel.

4.2 Multicast communication

The $1 \times N$ multicast communication involves message from a MSS destined to a set of MHs. Though the procedure seems to be similar to that of unicast mobile destination call model, there are a few significant differences. They are:

- (1) Instead of paging to a particular MH, the paging message is sent to a group.
- (2) Each MH should store the list of groups subscribed in order to respond to the paging message.
- (3) One channel is assigned for all group members, a MSS does not wait for any acknowledgement from all the MHs and thereby providing only a best-effort service.
- (4) If any of the subscribed MHs misses at least one frame after receiving the channel assignment message, the MH does not further listen to the multicast channel.

This is because, the MH may have missed the channel release message and the channel may no longer be valid. Hence, the MH disconnects itself from the channel.

4.2.1 Multicast channel establishment

The multicast channel establishment involves the following steps:

- (1) $MSS \rightarrow MH$: Schedule paging message to the group (using paging channel, PCH).
- (2) $MSS \rightarrow MH$: Channel assigned [using the next access grant channel (AGCH) after PCH for multicast communication].
- (3) $MSS \rightarrow MH$: One-way multicast communication (using traffic channel assigned from AGCH) takes place and all MHs receive the messages from the MSS.

4.3 Multicast channel release

The multicast channel release involves the following steps.

- (1) $MSS \rightarrow MH$: Channel release (using fast authoritative control channel).
- (2) $MSS \rightarrow MH$: Physical channel release.

5. The EOMP protocol

5.1 Overview

In this section, we describe the various aspects of our protocol viz. message transmission from a MH, totally ordered exactly once message delivery and handoff procedures. Various actions taken by MH, MSS and coordinators are explained in Sections 5.2.1–5.2.3. The detailed specification of the protocol in the form of pseudocode is given in Appendix A. Appendix B gives the formal proof of correctness.

5.1.1 Notations

We use CAPS to denote the message tags that identify the type of the message. We use $TAG\{m, n\}$ to refer to a message of type TAG with two fields m and n .

5.1.2 Message transmission from MHs

When the MH wants to multicast a message, the MH transmits the multicast request message (MCASTREQ) to the MSS. The MH retransmits the MCASTREQ message until an acknowledgement (MCASTACK) is received. Only after an MCASTACK is received, the MH starts transmitting the next message. This procedure is used to guarantee a reliable transmission of the message from MH to its coordinator. If the coordinator sends MCASTACK and if MCASTACK gets lost, then the MH times out and retransmits MCASTREQ. Now, the coordinator may receive a duplicate MCASTREQ. To detect the duplicate MCASTREQ, the coordinator needs to distinguish between the previous message and the next message. For this purpose, a one-bit sequence number ($t \times Seq$) is assigned to every MCASTREQ generated by a MH. Similarly, each coordinator maintains a one-bit sequence number ($r \times Seq[H]$) of the next MCASTREQ it expects to receive from each MH H . If the $r \times Seq[H]$ is not same as the $t \times Seq$ of the MCASTREQ received by the coordinator from MH H , then the message is discarded as duplicate. This procedure is similar to the stop-and-go protocol of [23].

5.1.3 Totally ordered exactly once message delivery

Every message multicast to the group is assigned a global unique sequence number. Each MH stores the sequence number of the last message delivered to the application, for each group G , in $lastSeqRecv[G]$. Similarly, every coordinator stores the last sequence number of the message multicast, to each group G , in $seqSent[G]$. Exactly once delivery of the messages is guaranteed by having these two sequence numbers and ensuring the following conditions:

- (1) The coordinators will never retransmit messages to the MH H , for the group G , with sequence numbers less than or equal to $lastSeqRecv[G]$.
- (2) The MH always delivers the messages to the application in the order of their sequence numbers and buffers the messages arriving out of order.

To ensure total ordering of messages, one coordinator acts as total ordering coordinator (TOC) and assigns sequence numbers to all the messages and multicasts them to all other coordinators. As the wired network is FIFO, every coordinator, and subsequently every MSS, receives the messages in the same order, thereby ensuring total ordering of messages.

5.1.4 Handoff procedure

A Handoff procedure is triggered by the MH by sending a GREET message to the MSS, whenever MH discovers that it has entered a new cell (known from the periodic beacon message sent by the MSS). The MH periodically retransmits GREET message until it receives a GREETACK message. The handoff overhead in the wired network (apart from the GREET and GREETACK messages) is a CANCEL message sent from the newMSS to the oldMSS (when MH switches cell from oldMSS to that of newMSS). The purpose of the CANCEL message is to cancel any retransmissions scheduled for MH at the oldMSS after the MH has left its cell. This is needed because if the CANCEL message is not sent and if the MH keeps oscillating between the cells, the MH may receive duplicates. One such scenario is shown in Figure 2. If a MH sends NACK at the old cell and moves to a new cell and sends a NACK (GREET has $lastSeqRecv$, which acts as an implicit NACK for the messages MH has not received). Then, if the MH moves back to the old cell and gets

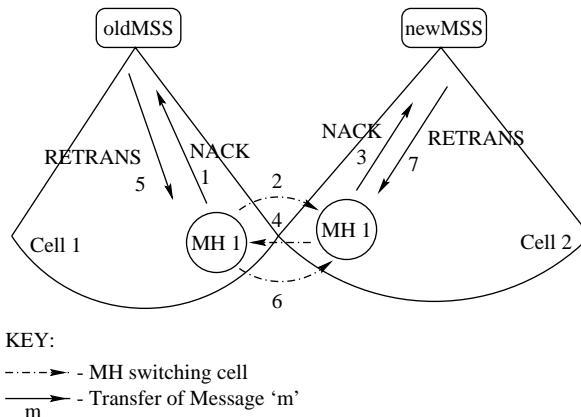


Figure 2. Worst case scenario – avoided by CANCEL message between MSS.

the RETRANS message and then, quickly switches again to the new cell and may receive the same RETRANS there too. To avoid such scenarios, cancel messages are used.

5.2 Protocol description

Figure 3 gives a pictorial description of various actions taken by the protocol to multicast a message from a MH.

5.2.1 Actions at MH

The steps taken at each MH are as follows:

- (1) Whenever an application wants to multicast a message m to group G and if it is currently not waiting for any MCASTACK, the MH sends MCASTREQ $\{m,G\}$ to the MSS and sets a timer waiting for MCASTACK.
 - (a) If timed out, the MH retransmits MCASTREQ $\{m,G\}$.
 - (b) If MCASTACK $\{m\}$ is received, the MH cancels the timer and is ready to accept the next MCASTREQ request.
- (2) Whenever a MH enters a new cell, the MH sends a GREET $\{myMSS, myCoord, lastSeqRecv[\]\}$ message to the MSS and sets a timer waiting for GREETACK. myMSS and myCoord refers to its respective MSS and the coordinator before cell switching.
 - (a) If timed out, the MH retransmits GREET $\{myMSS, myCoord, lastSeqRecv[\]\}$.

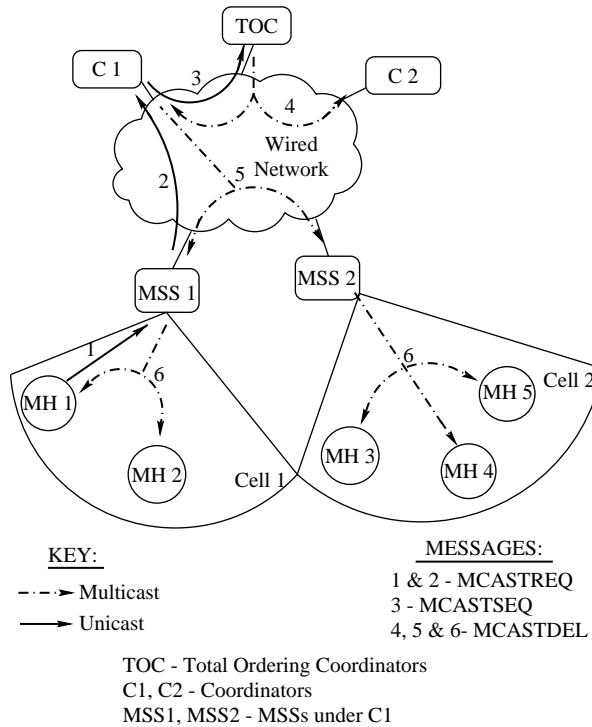


Figure 3. Protocol description.

- (b) If GREETACK{*newCoord*} is received, the MH cancels the timer and sets myMSS and myCoord to the new MSS and coordinator, respectively.
- (3) Whenever an application wants to join (or leave) a group G , the MH sends JOIN{ G } (or LEAVE{ G }) message to the MSS and sets a timer waiting for JOINACK (or LEAVEACK).
 - (a) If timed out, the MH retransmits JOIN{ G } (or LEAVE{ G }).
 - (b) If JOINACK (or LEAVEACK) is received, the MH cancels the timer.
- (4) Whenever a MH receives MCASTDEL{*seq,m,G*} (or RETRANS{*seq,m,G*}) from MSS, the MH buffers the message m .
 - (a) If arrived in order, the MH delivers all the consecutive messages from the buffer to the application in the increasing order of their sequence numbers and updates lastSeqRecv[] to reflect the last sequence number delivered to the application.
 - (b) If arrived out of order, the MH sends NACK{lastSeqRecv[] } to the MSS.

5.2.2 Actions at MSS

MSSs are designed as stateless. The MSSs are responsible only for forwarding messages between MHs and coordinators and are not involved in any decisions. The MSS has a cache for performance benefits and it does not affect the correctness as a MSS fails only by crashing. Further, along with every message forwarded to the coordinator, the MSS also appends the originating MH identifier.

The steps taken at each MSS are as follows:

- (1) Upon receiving MCASTREQ (or JOIN or LEAVE) messages from the MH, the MSS forwards them to the coordinator.
- (2) Upon receiving GREET message from the MH H , the MSS forwards the message to the coordinator and sends CANCEL{ H } to the old MSS (known from the first field of the GREET message).
- (3) Upon receiving NACK from the MH
 - (a) If the needed message m is in its cache, the MSS sends RETRANS{*seq,m,G*} to the MH, where *seq* is the sequence number of m .
 - (b) Otherwise, the MSS forwards the message to the coordinator and sets retransmission flag (if MH lags behind) to forward the subsequent RETRANS message from the coordinator.
- (4) Upon receiving GREETACK (or JOINACK or LEAVEACK or MCASTACK) from the coordinator, the MSS forwards the message to the MH.
- (5) Upon receiving RETRANS{*seq,m,G,H*} from the coordinator, if the retransmissions for MH H is not cancelled, then RETRANS{*seq,m,G*} is forwarded to H . Otherwise, the RETRANS is discarded.
- (6) Upon receiving CANCEL message from a MSS, it clears the retransmission flag to cancel any further retransmissions.

5.2.3 Actions by the coordinator

The steps taken by each coordinator are as follows:

- (1) Upon receiving MCASTREQ{ m,G }, the coordinator transmits MCASTACK destined to the originating MH H .

- (a) If MCASTREQ is a duplicate, MCASTREQ is discarded.
- (b) Otherwise, the $r \times \text{Seq}[H]$ is updated and $\text{SYNCSEQ}\{r \times \text{Seq}[H], H\}$ is sent to all other coordinators and a corresponding $\text{MCASTSEQ}\{m, G\}$ (for the MCASTREQ) message is sent to the TOC.
- (2) Upon receiving MCASTSEQ by the TOC from a coordinator, the TOC assigns a sequence number seq to the message based on its group G and multicasts $\text{MCASTDEL}\{seq, m, G\}$ to all coordinators.
- (3) Upon receiving $\text{MCASTDEL}\{seq, m, G\}$ from the TOC, the coordinator adds $\langle seq, m, G \rangle$ to the BufferList and multicasts $\text{MCASTDEL}\{seq, m, G\}$ to its MSS.
- (4) Upon receiving a NACK originated at MH H , the coordinator updates $\text{hostAckEntry}[H]$ to $\text{lastSeqRecv}[]$ of MH H . Then, the coordinator sends $\text{RETRANS}\{seq, m, G, H\}$ to the MSS, where m is the next message with sequence number seq needed by H (from the BufferList).
- (5) Upon receiving $\text{GREET}\{\text{myMSS}, \text{myCoord}, \text{lastSeqRecv}[], H\}$ message from the MSS, the coordinator sends $\text{GREETACK}\{H\}$ to the MSS.

Every coordinator stores the $\text{lastSeqRecv}[]$ of each MH H in $\text{hostAckEntry}[H]$. The purpose of $\text{hostAckEntry}[]$ is to find the messages acknowledged by all MHs and to delete them from the BufferList. The main purpose of the BufferList is only for retransmission upon loss of messages to guarantee reliability. Hence, when all hosts have delivered a message, there is no need for storing it.

Further, periodically, every coordinator sends $\text{hostAckEntry}[]$ to other coordinators by means of SYNCACK message. Instead of sending the full $\text{hostAckEntry}[]$, only those entries that have been modified since last SYNCACK message are sent. Upon receipt of SYNCACK message, coordinators update their $\text{hostAckEntry}[]$. Once, any message has been acknowledged by all MHs, the corresponding message is removed from the BufferList. This distributed peer–peer model of exchanging acknowledgement information is similar to the HeartBeat Algorithm [3] and it ensures faster convergence of the global acknowledgement scenario and thereby, reduces the buffer space at the coordinators.

6. Simulation study

To evaluate the performance of our protocol, simulation and comparison with an existing protocol is needed. As existing simulators like NS [14] and GloMoSim [26] do not support our system model, we developed an object-oriented discrete event simulator in C++ similar to [2].

We simulated our protocol and RelM protocol [6], which are also an exactly once protocol and seems to address similar issues. The parameters used for simulation are given in Tables 1 and 2. These parameters are used for simulation in [2]. As protocol [2] does not provide an exactly once message delivery, it was not simulated. The cell permanency time in Table 2 refers to the average time for which a MH will be inside a cell.

Our simulator is designed to measure the following parameters:

- average end-to-end delay;
- queuing delay at coordinator and MSS; and
- message buffer size at the coordinator.

We define end-to-end message delay as the time elapsed from the instant at which a multicast message is generated at the sender to the instant at which the same message is delivered to a destination MH.

Table 1. System parameters.

Parameter	Value
Number of groups	4
Number of coordinators	4
Number of MSS	32
Wired bandwidth	100 Mbps
Wireless bandwidth	10 Mbps
Wired propagation delay	0.5 ms
Wireless propagation delay	0.0 ms
Message loss probability	0.001
NACK loss probability	0.00
Out-of-range probability	0.00
NACK transmission period	1 s
MCASTREQ timeout	25 ms
SYNACK timeout	500 ms

Table 2. Tunable parameters.

Parameter	Value
Number of MHs	512
Number of senders	4
Message rate	250 messages/s
Message size	256 bytes
Cell permanency time	1 s

6.1 Effect of message size

Figure 4 shows that the average message delay for our protocol is very less (approximately 0.33% of RelM) when compared to RelM. The main reason for this is the use of wireless multicasting by our protocol instead of unicasting as in RelM. Due to an unicast, average delay increases as messages can be delivered to the MHs only after the messages for the previous MHs (in the queue) are delivered. Moreover, an unicast causes the messages to spend more time at the queues of the MSS waiting for transmission to the MH. As the queuing delay fluctuates widely, so does average delay for RelM. This can also be seen in Section 6.4.

6.2 Scalability

In this section, we compare the scalability of our protocol and RelM with respect to number of receiver MHs. Figure 5 shows that the increase in delay for our protocol is very negligible compared to RelM, as the number of receivers increase. Hence, our protocol scales very well compared to RelM.

The main reasons for this behaviour are multicasting in wireless medium and the use of a NACK-based protocol. Firstly, due to wireless multicasting, only one message is sent per group in a cell thereby greatly reducing message overhead at the MSS. On the other hand, in the case of RelM, unicasting is used to send messages to the MH. So, the number of messages sent increases with increase in the number of MHs and hence does not scale with the number of MHs. Secondly, if each MH sends an acknowledgement (ack) for every message received, Ack Implosion [18] occurs at MSS (and hence, also at coordinator). As, RelM uses this approach, it does not scale. However, our protocol uses a NACK-based

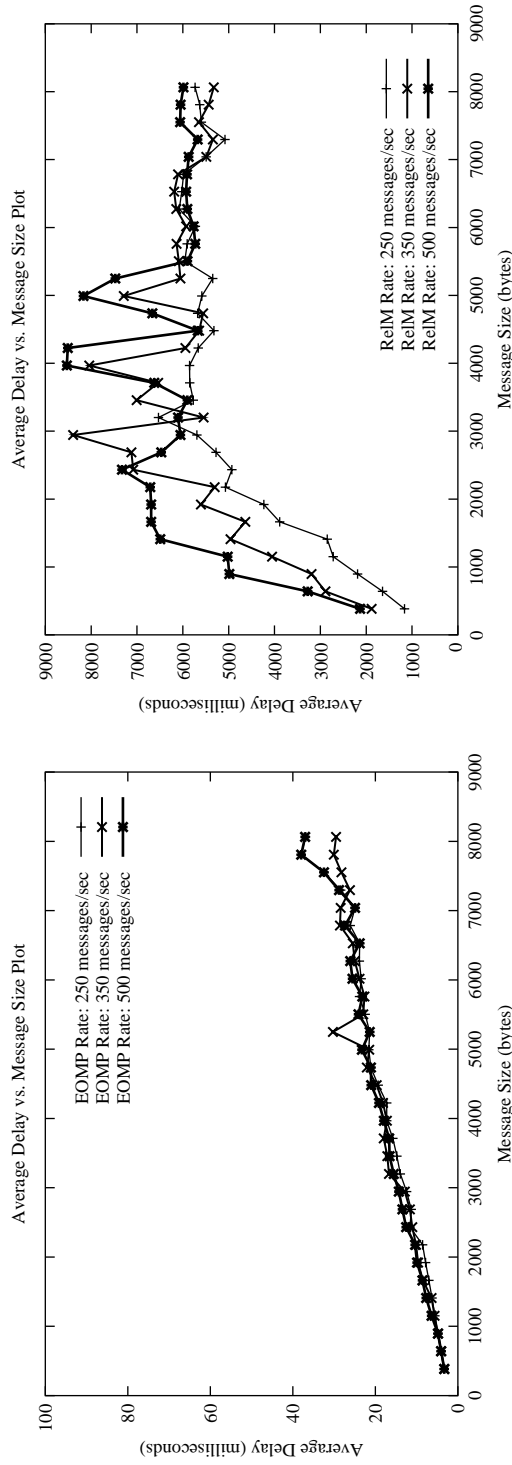


Figure 4. Effect of message size.

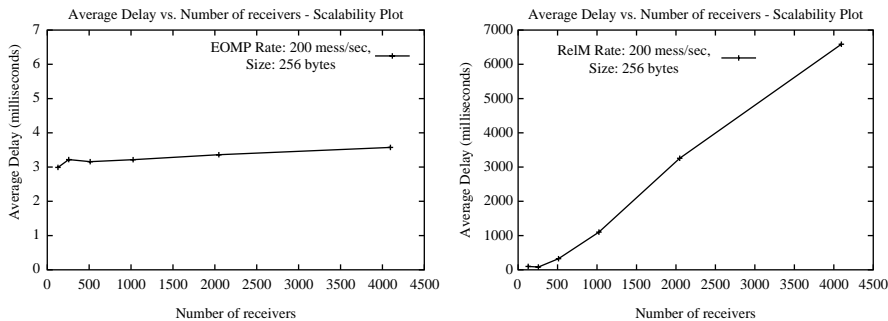


Figure 5. Scalability plot.

approach, where a NACK is sent only when a message arrives out of order at MH. Upon handoff, lastSeqRecv implicitly acknowledges all the messages received till then. Further, to guard against any lost NACK, periodically, NACK messages are sent by the MH. This NACK-based approach essentially eliminates the overhead due to Acks and hence scales well.

6.3 Effect of host mobility

Figure 6 shows that as cell permanency time decreases (i.e. as the number of cell switches per second increases), our protocol shows a stable behaviour compared to significant increase in delay for ReIM.

The reasons for this behaviour are the total ordering of messages and minimal handoff. As our protocol provides total ordering of messages, the last sequence number received by the MH corresponds to the same message at all coordinators and MSSs. Hence, there is no need for undelivered messages from the oldMSS to be forwarded to the newMSS, when MH moves from the cell of the oldMSS to that of the newMSS. This results in a very minimal message overhead at the wired network upon MH switching its cell. Only an extra CANCEL message is sent in our protocol. On the other hand, in ReIM, all the undelivered messages are forwarded by the oldMSS to the newMSS (via the coordinators of the oldMSS and the newMSS) upon every cell switching. Hence, effect of mobility is very marginal in our protocol.

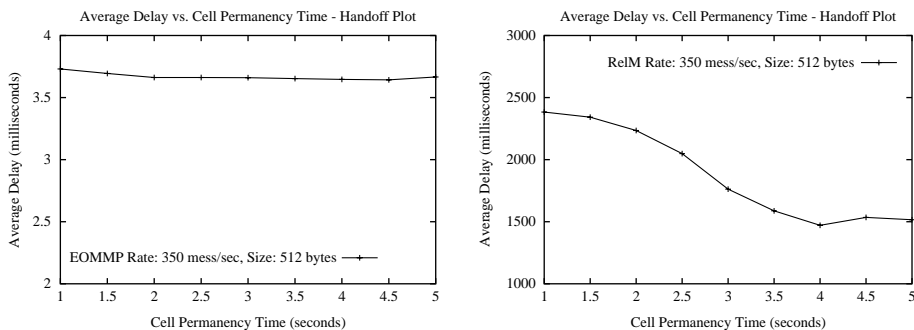


Figure 6. Mobility plot.

Table 3. Role of queuing delay.

Message size (bytes)	Average EOMP delay (ms)	EOMP TOC queue delay	EOMP Coord queue delay	EOMP MSS queue delay	Average RelM delay (ms)	RelM Coord queue delay	RelM MSS queue delay
256	3.33	0.004	0.29	0.39	576.81	155.91	200.01
512	5.26	0.006	0.29	0.65	1533.76	186.51	468.47
768	5.72	0.004	0.32	0.88	2358.58	351.83	876.42
1024	6.34	0.01	0.39	1.21	2658.29	303.30	1031.54
1280	11.16	0.01	0.38	1.42	3187.35	307.93	1324.92
1536	11.68	0.02	0.41	1.68	3595.71	552.88	1519.03
1792	12.47	0.02	0.48	1.84	4426.80	549.25	2024.69
2048	13.33	0.22	0.51	2.12	4879.77	550.05	2286.9

6.4 Role of queuing delay

We made one SH to act as just the TOC alone so that the delay for total ordering alone can be captured. To ensure total ordering of messages, every message is allocated sequence number by one TOC. Due to this, intuitively, the TOC must be the bottleneck. But Table 3 shows that the queuing delay at the TOC is not a major bottleneck. The bottleneck for both protocols seems to be the MSS. This is due to the significant mismatch between the bandwidths of the wired and the wireless media. Any node that interfaces two different network media with significantly different bandwidths will be a bottleneck. Similar results can also be seen in [2].

By comparing our protocol with RelM, we infer that the queuing delay constitutes a major portion of the average delay for RelM. In RelM, high coordinator queuing delay is primarily due to handoff messages and high MSS queuing delay is because of an unicast in a wireless medium. Moreover, due to an acknowledgement-based protocol also, the message traffic in the wired network is very high.

Since these bottlenecks are eliminated in our protocol, queuing delay does not contribute significantly to the average end-to-end delay and hence, the load on the wired network is very low even with the total ordering of messages.

Further, if the queuing delay contributes to the major portion of the average delay of a protocol, then it does not scale well. This can also be verified from Section 6.2.

6.5 Buffer utilisation

Figure 7 shows the buffer utilisation by the coordinators with respect to the message size. The decreasing trend for both the protocols is due to that at a given message rate, the number of messages generated for a given time decreases with the increasing message size. As the number of messages transmitted decreases, the average buffer usage also decreases. From Figure 7, it is very clear that our protocol uses less buffer space at the coordinators. The main reason for this is the use of a different mechanism for the removal of buffers for fully acknowledged messages.

In RelM, the deletion of a message from the buffer is decided only by the source of the message. Each coordinator buffers the message and waits for all MHs under it to acknowledge the message and only then, the coordinator sends DELIVERED message to the source of multicast. However, as the MHs move very quickly, a series of messages are passed between the coordinators to transfer any undelivered messages to the MHs. Only after ensuring that every MH under the coordinator has either acknowledged or moved to a new coordinator, the coordinator sends DELIVERED message to the source. If the source

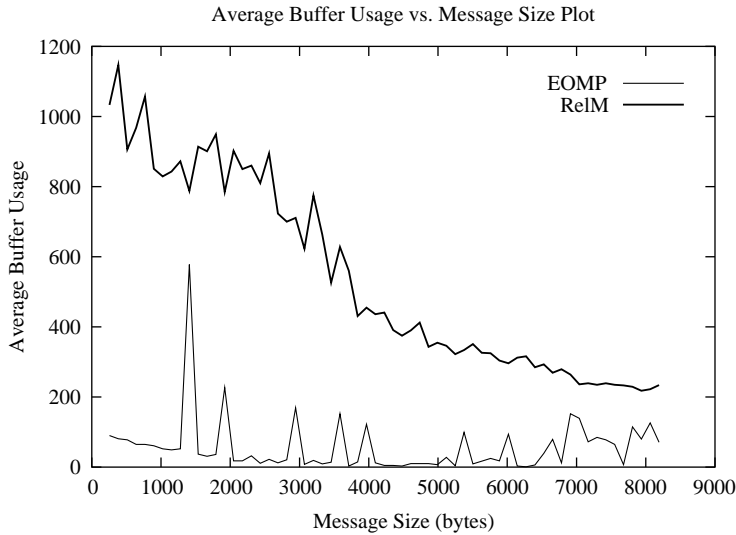


Figure 7. Buffer utilisation.

receives DELIVERED message from all coordinators, it sends a DISCARD message to all the coordinators. Such sequence of message transfers is needed for every message to be deleted from the buffer. This is time consuming.

In our protocol, we use a distributed peer-to-peer communication model to get the global acknowledgement information regarding each MH by every coordinator. This is ensured by the periodic exchange of SYNCACK messages and every coordinator can now independently delete the buffers of the fully acknowledged messages. Due to the use of a peer-to-peer model, every coordinator gets the global acknowledgement information within a few SYNCACK messages itself. Hence, our approach is host oriented instead of being message oriented. Further, one message regarding the lastSeqRecv[] of a MH is enough to indicate all the messages received by the MH. This causes the average buffer used by the coordinators to be low.

7. Case study 1: a distributed collaborative application

To establish the necessity for a reliable multicast protocol with totally ordered message delivery for distributed applications over mobile systems, we have implemented a distributed collaborative text editor over EOMP.

Distributed collaboration allows geographically dispersed users to closely cooperate with each other to solve a problem. Generally, collaborative applications involve a common workspace shared among a set of users, where modifications done by one can be viewed by everyone. This creates a need for close communication among the processes.

A distributed collaborative text editor application was developed for the case study due to the following reasons:

- (1) Collaborative text editor involves high message traffic and reliable multicast provides an efficient mechanism for structuring such communication.
- (2) Concurrency control in collaborative applications can be elegantly solved if the lower layer itself provides total ordering of messages.

- (3) The need to keep the application simple and hide the physical mobility and disconnections from the application.

The application was developed over our EOMP simulator. The text editor was written in Java and the user interface was designed using Java Swing. The text editor communicates with the EOMP simulator using sockets. In EOMP simulator, each MH is an object and has `transmit()` and `deliver()` methods for the application to transmit messages, and for the messages to be delivered to the application, respectively. Our text editor application writes into a socket for the EOMP simulator to read from for `transmit()`. For `deliver()`, EOMP simulator writes into a socket which is read by the application and displayed. Figure 8 shows the user interface of the text editor.

The application can edit one line at a time and after modification, pressing a *go* button causes the text editor to trigger a transaction. The transaction is sent to the MSS and later multicast to other hosts and executed at all MHs. The user can optionally disconnect from the session and reconnect later by the use of *disconnect* and *connect* buttons to force disconnected operations. Once the user presses the *connect* button, an EOMP ensures that the user is up-to-date. Alternatively, an EOMP at the lower layer itself simulates mobility and disconnections and these are hidden from the application.

8. Case study 2: distributed image rendering application

As mobile devices have become ubiquitous, computing power is present everywhere. This case study addresses the issue of distributed processing among MHs, by utilising the idle computing power of the devices. Distributed processing applications for mobile systems such as scientific applications to monitor seismic activities and other weather phenomenon such as tornadoes are discussed in [27].

Image rendering refers to the process by which the pixel values and the texture of the image are calculated and assigned to the respective pixel positions of the image [25]. Typical inputs to the image rendering application are wire-frame model of the image, binary data got from X-ray scan devices such as CT scan.

The characteristics of the image rendering application are as follows:

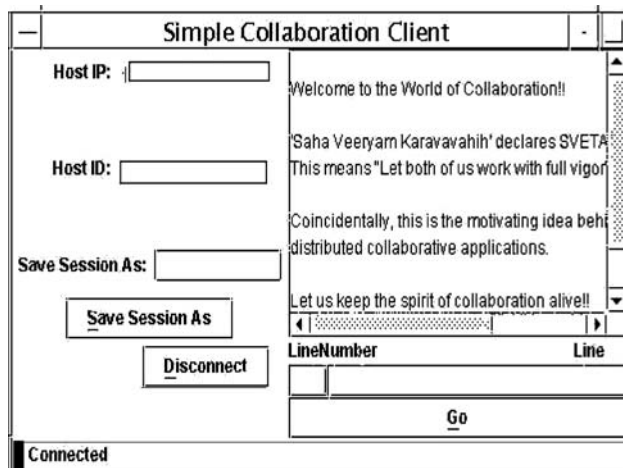


Figure 8. User interface of a collaborative text editor.

- (1) very large datasets;
- (2) computationally intensive; and
- (3) time consuming.

These characteristics motivate the need for distributed processing for image rendering applications.

Further, to render any frame of the image, the complete dataset is needed. Reliable multicast provides an efficient mechanism to transfer the datasets among the computing entities. As an EOMP provides a reliable multicast primitive with TOMD, the distributed image rendering application was developed over an EOMP. This section discusses the details of the developed distributed image rendering application.

8.1 System model

Each MH has a client and a daemon. The clients are used to submit tasks to the MHs for distributed processing. The daemons are the computing entities at the MHs that execute a part of the submitted task concurrently with other daemons. Both the client and daemon run over an EOMP and use the socket abstraction of an EOMP simulator. The system architecture is shown in Figure 9.

Each daemon keeps track of the total number of daemons (N) currently present in the group. Further, each daemon has a unique membership identifier (called ID, ranging from 0 to $N - 1$). MHs intended to participate in distributed processing subscribe to a common group. Whenever a MH wants to participate in distributed processing, the daemon at the MH joins the group and the MH will now receive tasks for distributed processing. Similarly, whenever a MH does not want to participate in distributed processing, it leaves the group and the MH will not receive further tasks for distributed processing.

8.2 Distributed image rendering

The key aspects of the distributed image rendering application are submission of a task for distributed processing and dynamic load balancing.

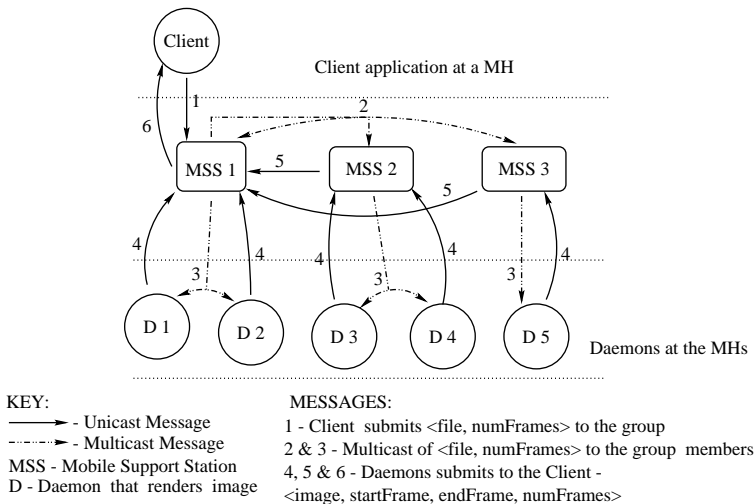


Figure 9. Distributed image rendering architecture for mobile systems.

8.2.1 Task submission

When a client wants to submit a task, it multicasts the task to the group. As the dataset is very large, the multicast provides an efficient mechanism for transferring the data to the daemons for processing. Each daemon on receiving the task, independently splits the task based on its ID and N . For example, frames having frame number ' f ' such that $\text{mod}(f, N) = \text{ID}$ are rendered by the daemon with identifier ID. When a daemon completes its part, the daemon sends the result back to the destination host.

8.2.2 Dynamic load balancing

Till now, each MH has a client and a daemon. When the MH does not want to participate in distributed processing (due to its heavy load), the daemon leaves the group. Before leaving, the daemon multicasts a Decrement (ID) message to the group. The Decrement (ID) message is an indication that daemon ID has left the group. So, each daemon decrements the total number of daemons and decrements its ID, if its ID is more than the ID of the leaving daemon. The leaving daemon sets its ID to -1 .

The need for multicasting the Decrement (ID) message is twofold. First, as every entity keeps track of the total number of computing entities currently subscribed to the group, each entity in the group must be informed about leaving of any entity from the group. Second, in a system of N entities, the identifiers must be contiguous and must always be in the range from 0 to $N - 1$. Hence, when an entity with an identifier i leaves the group, total computing entities become $N - 1$ and all entities having identifiers more than i decrement by 1.

Further, as each MH (or even SH) may have different capabilities (such as memory) and different processing power, it is essential to allocate tasks to the MHs based on their capabilities and processing power. To incorporate this, each MH is allocated an integer called Horse Power (HP). HP refers to the computational capability of the MH. When a MH has a HP ' h ', then h computing entities are allocated to the MH. For example, if MHs A and B have h_1 and h_2 as their respective HPs, then the time taken by A to compute h_1 amount of a task is approximately equal to the time taken by B to compute h_1 amount of the same task. Now, with HP incorporated into the system, N stands for the total number of computing entities in the system and ID refers to the identifier of that computing entity.

Dynamic load balancing [21] is done by maintaining two thresholds viz., upper threshold (UT) and lower threshold (LT) at the MH. These thresholds are shared by all the computing entities within a MH. This can be achieved by creating the computing entities as threads of the MH. When the load on the MH is greater than UT, a computing entity on that MH leaves the group and increments UT and LT on that MH. Both UT and LT are incremented so that all entities do not leave the groups at the same time. Similarly, when the load on the MH becomes lesser than LT, a computing entity on that MH joins the group and decrements both UT and LT. The need for two thresholds UT and LT is to avoid oscillations of frequent join and leave.

Further, the load balancing mechanism used is non-preemptive [21] and hence migration of already running task is not done. This is due to the additional communication overhead involved in process migration.

8.3 Performance analysis

The CT scan data of a human head with skull partially removed to reveal the brain are taken as the input data. The CT scan data consist of 84 slices of 128×128 samples each. The final output frame is 256×256 pixel image. The daemons were run on SPARC

333 MHz and SPARC 500 MHz machines having 256 MB RAM. The daemons use an EOMP simulator for multicasting to other MHs. These daemons are the application on the MHs simulated by the EOMP simulator. The EOMP simulator provides socket abstraction to the applications on the MHs. Hence, the daemons run on the SPARC machines and communicate with the respective MHs of the simulator by sockets.

Table 4 shows the results when daemons were executed on SPARC 333 MHz under no load condition. Super-linear speedup is noticed when number of hosts is 2. Table 5 shows the results when SPARC 500 MHz machines were used to execute daemons under no load conditions. Super-linear speedup is noticed when the number of hosts are 2–4. The reason for super-linear speedup is because, during the entire computation of 360 frames, the full volume data (which is reasonably large – 1.37 MB) needs to be in memory. During this time, page faults occur and the computation time increases. The context switch time is very negligible as the experiment was conducted in no load conditions (when the load on the machines were only due to the daemons). When the task is split to two hosts, computation time is only for half the task. Hence, the data need to be in the memory only for a much lesser time. As the memory residence time reduces, the page fault overhead also decreases. Reduction in these overheads (such as page faults and context switches, if any) and by overlapping computation and communication across hosts, the communication delay seems to be nullified. Thus, super-linear speedup is achieved. But, as the number of hosts increases to eight, the communication delay seems to dominate. This is partly because, an EOMP simulator runs on a single host and is not distributed. So, a multicast is simulated by multiple unicasts. If multicast is done physically (or a distributed simulator is used), then the communication overheads can be reduced further.

9. Conclusions and future work

We have proposed and evaluated an EOMP, a protocol with totally ordered message delivery, for mobile systems. Our protocol provides an efficient communication primitive for distributed applications to be structured upon. Traditionally, distributed applications are difficult to develop. Debugging and maintaining these distributed applications are even more difficult. When distributed applications are developed for mobile systems, due to the severe constraints of the MHs, issues such as mobility management need to be addressed

Table 4. No load performance analysis for ultra SPARC 333 MHz.

Number of hosts	Time taken (s)	Speedup
1	422	
2	206	2.05
3	142	2.97
4	107	3.94

Table 5. No load performance analysis for ultra SPARC 500 MHz.

Number of hosts	Time taken (s)	Speedup
1	298	
2	146	2.04
3	99	3.01
4	74	4.03
8	41	7.27

by the applications. As a result, the applications become very complex. Developing, debugging and maintaining such applications become very challenging. A solution to this problem by using a layered decomposition model is proposed in this report. As mobility and disconnections are the characteristics of the system and not the application, the application must not be aware of mobility and disconnection. Hence, a lower layer needs to take care of such issues. Further, as an exactly once multicast is essential for distributed mobile systems as shown by the two case studies; an EOMP acts as an ideal building block for distributed mobile applications. Moreover, as an EOMP captures semantic requirements of many applications (such as transaction-based applications) at a much lower layer, the applications become simpler.

Real-time guarantees are not provided by an EOMP and are provided only by the application. Incorporation of quality of service guarantees at the lower layer is a real challenge. Tolerating faults at the MH, possibly by checkpointing, also needs to be addressed. A protocol similar to three phase commit [5] may solve the problem.

Notes

1. Email: vrd@cs.iitm.ernet.in
2. Email: djram@cs.iitm.ernet.in
3. Cell is a region under a MSS, where a MSS is capable of communicating with any MH under it.
4. In GSM systems, a channel number and a timeslot are assigned.

References

- [1] A. Acharya and B.R. Badrinath, *A framework for delivering multicast messages in networks with mobile hosts*, ACM/Baltzer Mob. Netw. Appl. 1 (1996), pp. 199–219.
- [2] G. Anastasi, A. Bartoli, and F. Spadoni, *A reliable multicast protocol for distributed mobile systems: Design and evaluation*, IEEE Trans. Parallel Distrib. Syst. 12 (2001), pp. 1009–1022.
- [3] G.R. Andrews, *Paradigms for process interaction in distributed programs*, ACM Comput. Surv. 23 (1991), pp. 49–90.
- [4] V. Aravamudhan, K. Ratnam, and S. Rangarajan, *An efficient multicast protocol for PCS networks*, ACM/Baltzer Mob. Netw. Appl. 2 (1998), pp. 333–344.
- [5] P.A. Bernstein and N. Goodman, *Concurrency control in distributed database systems*, ACM Comput. Surv. 13 (1981), pp. 185–221.
- [6] K. Brown and S. Singh, *RelM: Reliable multicast for mobile networks*, J. Comput. Commun. 21(16) (1998), pp. 1379–1400.
- [7] R. Chandra, V. Ramasubramanian, and K. Birman, *Anonymous gossip: Improving multicast reliability in mobile ad-hoc networks*, in *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS)*, 2001, pp. 275–283.
- [8] L. Harte and R. Levine, *Lectures on digital switching: Cellular and PCS*, Available at: <http://www.privateline.com/celpcslevine.pdf>, 2001.
- [9] T. Inoue and R. Kurebayashi, *Evaluating the impact of tunneling on multicast efficiency*, in *Asia-Pacific Conference on Communications*, 2005, pp. 254–258.
- [10] J. Kuri and S.K. Kasera, *Reliable multicast protocol in multi-access wireless LAN*, Wireless Netw. 7 (2001), pp. 359–369.
- [11] J.-R. Lai and W. Liao, *Mobile multicast with routing optimization for recipient mobility*, IEEE Trans. Consum. Electron. 47(1) (2001), pp. 199–206.
- [12] C.R. Lin and K.-M. Wang, *Scalable multicast protocol in IP-based mobile networks*, Wireless Netw. 8(1) (2002), pp. 27–36.
- [13] C. Livadas, I. Keidar, and N.A. Lynch, *Designing a caching-based reliable multicast protocol*, in *Proceedings of the International Conference on Dependable Systems and Networks (DSN'01)*, Gothenburg, Sweden, 2001.
- [14] S. McCanne and S. Floyd, *NS-network simulator*, 1995.
- [15] M.M. Mohamed and D. Janakiram, *Fault-tolerant exactly once reliable multicast protocol for distributed mobile systems*, in *Proceedings of the International Conference on Communication Systems and Networks (CSN 2003)*, Spain, 2003.

- [16] D. Pathirana and M. Kwon, *RODMRP: Resilient on-demand multicast routing protocol*, in *21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07)*, 2007.
- [17] C. Perkins, *IP Mobility Support*, RFC 2002, Network Working Group, 1996.
- [18] S. Pingali, D. Towsley, and J.F. Kurose, *A comparison of sender-initiated and receiver-initiated reliable multicast protocols*, IEEE J. Sel. Areas Commun. 15(3) (1997).
- [19] H. Rajabi, N. Nemat-Bakhsh, and N. Movahedinia, *Mobile multicast support using old foreign agent (MMOFA)*, in *Proceedings of the International Conference on Computer Information and Systems Science and Engineering, CISE-2007*, Bangkok, Thailand, 2007.
- [20] T.S. Rappaport, *Wireless Communications: Principles and Practice*, Prentice-Hall PTR, Upper Saddle River, NJ, 1996.
- [21] M. Singhal and N.G. Shivaratri, *Advanced Concepts in Operating Systems*, McGraw-Hill Inc, New York, 1994.
- [22] M.T. Sun, L. Huang, A. Arora, and T.H. Lai, *Reliable MAC layer multicast in IEEE 802.11 wireless networks*, in *Proceedings of the IEEE ICPP 2002*, Vancouver, BC, Canada, 2001.
- [23] A.S. Tanenbaum, *Computer Networks*, 3rd ed., Prentice-Hall of India Pvt. Ltd, New Delhi, 1997.
- [24] G. Wang, J. Cao, and K.C.C. Chan, *A reliable totally-ordered group multicast protocol for mobile internet*, in *International Conference on Parallel Processing Workshops (ICPPW'04)*, 2004.
- [25] A. Watt and M. Watt, *Advanced Animation and Rendering Techniques: Theory and Practise*, Addison-Wesley Publishing Company, Reading, MA, 1992.
- [26] X. Zeng, R. Bagrodia, and M. Gerla, *Glomosim: A library for parallel simulation of large-scale wireless networks*, in *Workshop on Parallel and Distributed Simulation*, 1998, pp. 154–161.
- [27] H. Zheng, R. Buyya, and S. Bhattacharya, *Mobile cluster computing and timeliness issues*, J. Informatica (Slovenia) 23(1) (1999).

Appendix A. Detailed protocol specification

The protocol is described in the form of self-explanatory pseudocode for various actions viz. message transmissions, handoffs and dynamic join/leave of groups by hosts.

A.1 Notations

We use CAPS to denote the message tags that identify the type of the message. This is needed as we use a stateless approach. We denote A unicasting message M to B by

$$A - \langle M \rangle \rightarrow B.$$

We denote A multicasting message M to set of nodes Bi by

$$A = \langle M \rangle \Rightarrow B_i.$$

We use C to refer to the coordinators, M to refer to MSS, H to refer to MH, *m* to refer to the message, mseq to refer to the sequence number assigned to message *m* and G to refer to the group number.

A.2 Message transmission

The message tags used for message transmission are as follows:

- MCASTREQ – Request for multicast (from $H \rightarrow M$ & $M \rightarrow C$).
- MCASTACK – Acknowledge the receipt of MCASTREQ (from $C \rightarrow M$ & $M \rightarrow H$).
- MCASTSEQ – Allocate a sequence number for the message (from $C \rightarrow TOC$).
- MCASTDEL – Deliver the multicast message (from the $TOC \rightarrow C$).
- SYNCSEQ – Synchronise the $T \times Seq$ of H with $R \times Seq[H]$.
- RETRANS – Retransmit message (from $H \rightarrow M$ & $M \rightarrow C$).
- NACK – Negative Acknowledgment (from $H \rightarrow M$ & $M \rightarrow C$).

The steps involved in message transmission are:

- (1) $H - \langle \text{MCASTREQ}, m, G, \text{LastSeqRecv}[], T \times \text{Seq} \rangle \rightarrow M$
- (2) $M - \langle \text{MCASTREQ}, m, G, H, \text{LastSeqRecv}[], \text{SeqSent}[], T \times \text{Seq} \rangle \rightarrow C$
and M sets `mcastFlag` for H and
if H lags behind, M sets `retransNeeded` flag for H
- (3) $C - \langle \text{MCASTACK}, T \times \text{Seq}, G, H \rangle \rightarrow M$
- (4) If `mcastFlag` is set,
then
 $M - \langle \text{MCASTACK}, T \times \text{Seq}, G \rangle \rightarrow H$
- (5) H updates $T \times \text{Seq}$
- (6) If $R \times \text{Seq}[H] < T \times \text{Seq}$,
then
 - (a) $C = \langle \text{SYNCSEQ}, T \times \text{Seq}, H \rangle \Rightarrow C_j, C_j' \setminus C$
 - else
Discard the message and do not multicast m as it is a duplicate
- (7) C updates `HostAckEntry[H]`-based on `LastSeqRecv[]` and $R \times \text{Seq}[H]$ -based on $T \times \text{Seq}$
- (8) If (H lags behind M),
/* by comparing `LastSeqRecv[]` & `SeqSent[]` of M */
 - (a) $C - \langle \text{RETRANS}, \text{msg}, \text{msgseq}, G, H \rangle \rightarrow M$
/* msg – next message needed by H */
 - (b) If `retransNeeded` flag is set,
then
 - i. $M - \langle \text{RETRANS}, \text{msg}, \text{msgseq}, G \rangle \rightarrow H$
 - ii. M clears the `retransNeeded` flag
 - else
 - i. break out of if(H lags behind M)
 - (c) H updates `LastSeqRecv[]` and delivers messages upto `LastSeqRecv[]`.
- (9) $C - \langle \text{MCASTSEQ}, m, G \rangle \rightarrow \text{TOC}$
- (10) TOC assigns sequence number `mseq` to m
- (11) $\text{TOC} = \langle \text{MCASTDEL}, m, \text{mseq}, G \rangle \Rightarrow C_i, C_i' \setminus \text{TOC}$
- (12) C_i updates `SeqSent[G] = mseq` and buffers $\langle m, \text{mseq}, G \rangle$ in `BufferList`
- (13) $C_i = \langle \text{MCASTDEL}, m, \text{mseq}, G \rangle \Rightarrow M_{ij}$
/* M_{ij} - MSS_j under *coordinator* _{i} */
- (14) $M_{ij} = \langle \text{MCASTDEL}, m, \text{mseq}, G \rangle \Rightarrow H_k$
/* by wireless multicasting */
- (15) At each host H , If m is out of order,
then
 - (a) $H - \langle \text{NACK}, \text{LastSeqRecv}[] \rangle \rightarrow M$
 - (b) $M - \langle \text{NACK}, \text{LastSeqRecv}[], H \rangle \rightarrow C$
 - (c) M sets `retransNeeded` flag if H lags behind
 - (d) C updates `HostAckEntry[H]` based on `LastSeqRecv[]`
 - (e) $C - \langle \text{RETRANS}, \text{msg}, \text{msgSeq}, G, H \rangle \rightarrow M$
/* msg – next message needed by H */
 - (f) If `retransNeeded` flag is set,
then
 - i. $M - \langle \text{RETRANS}, \text{msg}, \text{msgSeq}, G \rangle \rightarrow H$
 - ii. M clears the `retransNeeded` flag
 - else
 - i. break out of if(H lags behind M)
 - (g) H updates `LastSeqRecv[]` and delivers messages upto `LastSeqRecv[]`
- (16) H updates `LastSeqRecv[]` and delivers messages upto `LastSeqRecv[]`

Apart from these messages, periodically, MH sends `NACK` to guard against lost `NACKs`. Also, each coordinator sends `SYNCACK` message with `lastSeqRecv` of the hosts, which have newly sent `NACK` since last `SYNCACK` message. `SYNCACK` message sent to all coordinators, enables the coordinators to get a global picture of fully acknowledged messages and to remove those messages from the `BufferList`.

A.3 Handoff

The message tags used in handoff procedure are:

- GREET – Request for Handoff (from H → M & M → C).
- GREETACK – Acknowledge the receipt of GREET (from C → M & M → H).
- RETRANS – Retransmit message (from H → M & M → C).
- CANCEL – Cancel the scheduled message (re)transmissions for the MH, due to the MH moving out of the region.

The steps involved in the handoff procedure are:

- (1) H – $\langle \text{GREET, MyMSS, MyCoord, LastSeqRecv} \rangle \rightarrow M'$
- (2) M' creates a retransNeeded flag for H (if already not present) and clears it
- (3) M' – $\langle \text{CANCEL, H} \rangle \rightarrow \text{MyMSS}$
- (4) MyMSS upon receiving cancel for H, clears retransNeeded flag and mcstFlag
- (5) If (M' lags behind H),
 - then
 - (a) M' buffers the greet request until SeqSent[] of MSS catches up with LastSeqRecv[] of MH
 - else,
 - (a) If (H lags behind M'),
 - then
 - i. it sets the retransNeeded flag for H
- (6) M' – $\langle \text{GREET, H, LastSeqRecv} \rangle, \text{SeqSent} \rangle, \text{MyCoord} \rangle \rightarrow C$
 /* SeqSent[] corresponds to M' */
- (7) C updates the HostAckEntry[H]-based on LastSeqRecv[]
- (8) if (H lags behind M'),
 - then /* by comparing LastSeqRecv[] & SeqSent[] of M' */
 - (a) C – $\langle \text{RETRANS, msg, msgSeq, G, H} \rangle \rightarrow M'$ /* msg – next message needed by H */
 - (b) If retransNeeded flag is set,
 - then
 - i. M' – $\langle \text{RETRANS, msg, msgseq, G} \rangle \rightarrow H$
 - ii. M' clears the retransNeeded flag
 - else
 - i. break out of if (H lags behind M')
- (8) H updates LastSeqRecv[] and delivers the msg to the upper layer
- (9) C – $\langle \text{GREETACK, H} \rangle \rightarrow M'$
- (10) M' – $\langle \text{GREETACK, SeqSent} \rangle, C \rangle \rightarrow H$
- (11) H sets MyMSS = M' , MyCoord = C and updates LastSeqRecv[].

A.4 Dynamic join/leave of groups by MHs

The message tags used in dynamic join/leave procedure are:

- JOIN – Request for Join (from H → M & M → C).
- LEAVE – Request for Leave (from H → M & M → C).
- JOINACK – Acknowledge the receipt of Join (from C → M & M → H).
- LEAVEACK – Acknowledge the receipt of Leave (from C → M & M → H).
- INSERT – Send Join message to other coordinators (from C → C).
- DELETE – Send Leave message to other coordinators (from C → C).

The steps involved in JOIN procedure are:

- (1) H – $\langle \text{JOIN, G} \rangle \rightarrow M$
- (2) M – $\langle \text{JOIN, H, G} \rangle \rightarrow C$ and M becomes a member of G (if already not) to receive the multicast messages for G
- (3) If H is already not a member of G, then
 - (a) C adds H to MemberSet[G] and C becomes a member of G (if already not) to receive the multicast messages for G
 - (b) C – $\langle \text{INSERT, H, G} \rangle \rightarrow C_j, C_j \langle \rangle C$. Upon receiving $\langle \text{INSERT, H, G} \rangle$, C_j adds H to MemberSet[G].

- (4) $C \rightarrow (JOINACK, H) \rightarrow M$
- (5) $M \rightarrow (JOINACK, SeqSent[]) \rightarrow H$
- (6) M creates a retransNeeded flag for H (if already not present) and clears it
- (7) H sets $MyMSS = M$, $LastSeqRecv[] = SeqSent[]$

For procedure upon Leave, replace JOIN by LEAVE, INSERT by DELETE and JOINACK by LEAVEACK.

Appendix B. Proof of protocol correctness

LEMMA 1. The Protocol ensures that each MH receives atmost one copy of original multicast without MSS failures.

Proof. If the MH is stationary during multicast of the message in the system, it will receive the multicast at step (14) of Section A.2 and then the MH updates its sequence number to reflect the receipt of the multicast message. As the MH does not crash or behave maliciously, it will receive the original multicast message atmost once.

If the MH moves from the oldMSS to newMSS, there are four cases to explore.

- (1) The MH has received an original multicast in its old cell and the newMSS has not yet multicast the message. In such a situation, during the handoff procedure, the greet request is buffered at step (5) of Section A.3 till newMSS multicasts the message and therefore MH does not receive it again. Hence, MH receives the message exactly once.
- (2) The MH has not received the original multicast in its old cell and the newMSS has not yet multicast the message. In such a situation, oldMSS is yet to receive the message from the coordinator and as the MH does not lag behind oldMSS and condition (8) of Section A.3 is false, the coordinator does not retransmit and the newMSS eventually gets the multicast message from the coordinator and multicasts the message to its cell. Hence, the MH gets the multicast message atmost once.
- (3) The MH has received original multicast in its old cell and the newMSS has multicast the message. Under this situation, both the newMSS and MH are on par with each other and no retransmission is triggered in the new cell because condition (8) of Section A.3 is false. Hence, MH receives the message exactly once.
- (4) The MH has not received the original multicast (and hence, satisfies atmost once) in its old cell and the newMSS has multicast the message. In this case, it does not get the original multicast but as the MH lags behind newMSS, condition (8) of Section A.3 triggers a retransmission.

Note that any retransmission scheduled for MH at the old cell is cleared as soon as the MH moves to a different cell. \square

THEOREM 1. The Protocol ensures an exactly once message delivery without MSS failure.

Proof. From Lemma 1, it is clear that as long as the MH is stationary or satisfies case 1 or 3, the MH receives the original message exactly once and does not trigger retransmission for the message. Now, it is enough to show that from cases 2 and 4 also, it will receive the message exactly once.

In case 2, the message will be multicast in the near future at the new cell and if the host does not miss this message, it updates the $LastSeqRecv[]$ and gets the message during multicast itself. If the host moves to cell1 from cell2 and receives the message at cell1 and comes back to the cell2 before the message is multicast at cell2, the GREET message is buffered in step (5)a of Section A.3 and the MH does not receive the message again. If the MH does not receive at cell1, and comes back to cell2 again, it will either be in the same scenario or will move to case 4 depending upon if the cell2 has multicast the message or not.

In case 4, it triggers retransmission and sets the flag in step (5)(a)i of Section A.3. To avoid multiple retransmissions scheduled for the same host, earlier retransmissions, if any, are cleared upon CANCEL message. Hence, retransmission will be scheduled atmost at one site for each MH at any given time. Hence, as long as the MH does not fail and does not behave maliciously, it receives the message from *exactly one location*. \square

THEOREM 2. The protocol ensures an exactly once message delivery even if any MSS crashes at any time.

Proof. From Theorem 1, we have proved that if MSS does not fail, an exactly once message delivery is guaranteed. Further, we need to prove that even if any MSS crashes, an exactly once message delivery is guaranteed.

As MSS is *stateless* with no critical state and merely acts as a cache for coordinator, failure of any MSS at any time will be equivalent to all the MH under that cell becoming out-of-range. According to our assumption that MH will never be out-of-range forever and either the MH moves to a valid cell or the MSS is eventually replaced with a state consistent with the coordinator, all MHs receive messages exactly once. \square