JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 24, 573-583 (2008)

Short Paper

A Delay-Optimal Group Mutual Exclusion Algorithm for a Tree Network

VINAY MADENUR AND NEERAJ MITTAL+
Internet Services
Qualcomm, Inc.
San Diego, CA 92126, U.S.A.
E-mail: madenur.vinay@gmail.com
+*Department of Computer Science*
The University of Texas at Dallas
Richardson, TX 75083, U.S.A.
E-mail: neerajm@utdallas.edu

The group mutual exclusion problem is an extension of the traditional mutual exclusion problem in which every critical section is associated with a *type* or a *group*. Processes requesting critical sections of the same type can execute their critical sections concurrently. However, processes requesting critical sections of different types must execute their critical sections in a mutually exclusive manner.

We present an efficient distributed algorithm for solving the group mutual exclusion problem when processes are arranged in the form of a tree. Our algorithm is derived from Beauquier *et al.*'s group mutual exclusion algorithm for a tree network. The message complexity of our algorithm is at most $3h_{max}$, where h_{max} is the maximum height of the tree when rooted at any process. Its waiting time and synchronization delay, measured in terms of number of message hops, are at most $2h_{max}$ and h_{max}, respectively. Our algorithm has *optimal* synchronization delay for the class of tree network based algorithms for group mutual exclusion in which messages are only exchanged over the edges in the tree.

Our simulation experiments indicate that our algorithm outperforms Beauquier *et al.*'s group mutual exclusion algorithm by as much as 70% in some cases.

Keywords: distributed system, resource management, group mutual exclusion, tree network, optimal synchronization delay

1. INTRODUCTION

Mutual exclusion is one of the most fundamental problems in concurrent systems including distributed systems. In this problem, access to a shared resource (that is, execution of critical section) by different processes must be synchronized to ensure its integrity by allowing at most one process to access the resource at a time. Numerous solutions [1-5] and extensions [6-9] have been proposed to the basic mutual exclusion problem. Recently, Joung [10] proposed another extension to the basic mutual exclusion problem referred to as the *group mutual exclusion* problem. In the group mutual exclusion prob-

Received December 26, 2005; revised May 4 & September 20, 2006; accepted October 4, 2006.
Communicated by Tsan-sheng Hsu.

lem, every critical section is associated with a *type* or a *group*. Critical sections belonging to the same group can be executed concurrently while critical sections belonging to different groups must be executed in a mutually exclusive manner.

The readers/writers problem can be modeled as a special case of group mutual exclusion using $n + 1$ groups, where n denotes the number of processes. In this case, all *read* requests belong to the same group and *write* request by each process belongs to a different group. As another application of the problem, consider a CD-jukebox where data is stored on disks and only one disk can be loaded for access at a time [10]. In this example, when a disk is loaded, users that need data on the currently loaded disk can access the disk concurrently. While users that need data on different disks have to wait for the currently loaded disk to be unloaded.

Solutions for the group mutual exclusion problem have been proposed under both shared-memory and message-passing models. Solutions under shared-memory model can be found in [10-13]. In this paper, we investigate the group mutual exclusion problem under message-passing model. Under message-passing model, solutions for group mutual exclusion have been proposed for a fully connected network [14-18], ring network [19, 20] and tree network [21]. In a tree network, processes are assumed to be arranged in the form of a tree. The arrangement may correspond to a spanning tree of the underlying communication topology or may be just logical in nature. Typically, tree network based algorithms for group mutual exclusion have lower message complexity than other algorithms when the height of the tree is small (*e.g.*, $O(\log n)$).

Beauquier *et al*. present three group mutual exclusion algorithms for a tree network [21], namely GME_α, GME_β and GME_γ. GME_γ is an improvement over GME_β, which in turn is an improvement over GME_α. Their message complexities are at most $3(n - 1) + h_I$, $4h_I$ and $4h_{\max}$, respectively, where n is the number of processes in the system, h_I is the initial height of the tree, and $h_{\max}$ is the maximum height of the tree when rooted at any process. (Actually, $h_{\max}$ is same as the diameter of the tree. Also, $h_{\max} \leq 2h_I$.) Their waiting times in terms of number of message hops, measured when the system is lightly loaded, are at most $2h_I$, $2h_I$ and $2h_{\max}$, respectively. Finally, their synchronization delays in terms of number of message hops, measured when the system is heavily loaded, are at most $3h_I$, $3h_I$ and $3h_{\max}$, respectively. In GME_α and GME_β, the root of the tree remains fixed and is responsible for handling all requests for critical section generated in the system. As a result, processes closer to the root have to handle more messages than those that are farther away from the root. On the other hand, in GME_γ, there is no fixed root; root of the tree may change with time. Informally, the current root of the tree "manages" only those requests that are compatible with its own. (In fact, GME_γ can also be viewed as an extension of Raymond's tree-based algorithm for traditional mutual exclusion [5].)

In this paper, we present a more efficient algorithm for group mutual exclusion for a tree network. Our algorithm is derived from GME_γ. Specifically, we suggest modifications to GME_γ to significantly improve its performance. The algorithm obtained after the modifications has message complexity of at most $3h_{\max}$, waiting time of at most $2h_{\max}$ and synchronization delay of at most $h_{\max}$. Our simulation experiments indicate that our algorithm outperforms GME_γ with respect to all three metrics (message complexity, waiting time and system throughput). Specifically, for some of the parameter values we used in our experiments, it exhibited 35% lower message complexity, 40% lower waiting time and 70% higher system throughput than GME_γ.

The rest of the paper is organized as follows. We present our system model and formally describe the group mutual exclusion problem in section 2. Section 3 describes the three group mutual exclusion algorithms for a tree network by Beauquier *et al*. [21]. We describe our group mutual exclusion algorithm for a tree network in section 4 and also present our simulation results. Finally, we present our conclusions in section 5.

2. MODEL AND PROBLEM DEFINITION

2.1 System Model

We assume an asynchronous distributed system in which processes communicate by exchanging messages over a set of communication channels. In this paper, we assume that processes are connected in the form of a tree. Therefore a communication channel exists between two processes if and only if the two processes are neighbors in the tree. There is no global clock or shared memory. Processes are non-faulty and channels are reliable. Message delays are finite but may be unbounded. The channels are assumed to be first-in-first-out (FIFO).

2.2 The Group Mutual Exclusion Problem

The group mutual exclusion (GME) problem is an extension to the basic mutual exclusion problem. In this problem, a *type* or a *group* is associated with each critical section. Critical sections belonging to the same group can be executed concurrently while critical sections belonging to different groups must be executed in a mutually exclusive manner. The group mutual exclusion problem was first proposed by Joung in [10]. Any algorithm that solves the group mutual exclusion problem should satisfy the following properties:

- **group mutual exclusion (safety):** At any time, no two processes that have requested critical sections belonging to different groups are in their critical sections simultaneously.
- **starvation freedom (liveness):** A process requesting entry into its critical section should eventually be able to enter the critical section.

Observe that any algorithm that solves the traditional mutual exclusion satisfies the aforementioned properties. However, these algorithms are sub-optimal. Even if two requests for critical sections belong to the same group, a (traditional) mutual exclusion algorithm would force the requesting processes to execute their critical sections one-by-one in a mutually exclusive manner. In other words, no concurrency whatsoever is permitted. Therefore it is desirable that a group mutual exclusion algorithm should satisfy the following property as well:

- **concurrent entry (non-triviality):** If all requests are for critical sections belonging to the same group, then a requesting process should not be required to wait for entering its critical section until some other process has left its critical section.

With concurrent entry property, trivial solutions to group mutual exclusion problem

are ruled out. We assume that a process stays in its critical section for a finite but unbounded amount of time.

2.3 Complexity Measures

To measure the performance of a group mutual exclusion algorithm, we use the following metrics:

- *message complexity*: the number of messages exchanged per request for critical section.
- *message-size complexity*: the amount of data piggybacked on a message in terms of number of bits.
- *synchronization delay*: the time elapsed between when some process leaves its critical section and some other process can enter its critical section of different type.
- *waiting time*: the time elapsed between when a process issues a request for critical section and when it actually enters the critical section.
- *system throughput*: the number of critical section requests fulfilled per unit time.
- *concurrency*: the number of processes that are in their critical sections at the same time.

The first five metrics are used to evaluate the performance of a traditional mutual exclusion algorithm as well. The sixth metric is specific to a group mutual exclusion algorithm. We measure synchronization delay and waiting time in terms of number of message hops rather than in terms of time.

Message complexity and message-size complexity together capture the overhead imposed on the communication network by the group mutual exclusion algorithm at runtime. Synchronization delay is measured when the system is heavily loaded and a large number of processes are competing among themselves for accessing the resource. Intuitively, synchronization delay and concurrency measure the system throughput that can be achieved when the system is heavily loaded. The lower the synchronization delay and higher the concurrency, the higher is the system throughput. Waiting time captures the amount of time an application process has to wait for its request to be fulfilled. Waiting time is typically measured when the system is lightly loaded and, therefore, there is no contention for the resource.

3. BACKGROUND: BEAUQUIER *ET AL.*'S ALGORITHMS

Beauquier *et al*. present three algorithms for group mutual exclusion suitable for a tree network, namely GME_α, GME_β and GME_γ in [21]. All three algorithms use the notion of *session*. Informally, a session is initiated by the current root of the spanning tree and has a specific type. While a session is in progress, processes that have requested critical sections of that type can enter and execute their critical sections.

The main idea behind the three algorithms is as follows. Every request for a critical section is forwarded to the current root of the tree via parent edges with the following optimization. The request is forwarded until either (1) it reaches a process that is aware of a session in progress whose type is same as that of the request (that is, the request is *compatible* with the session), or (2) it reaches the root of the tree. In the former case, the

requesting process is invited to join the session in progress. In the latter case, the root enqueues the request in its queue (of outstanding requests) until a session of the same type as the request is initiated. Further, a process forwards at most one *pending* request for a critical section of any type to its parent. A session is managed using three waves: *open session wave*, *close session wave* and *completion wave*. Informally, the first wave initiates opening of a session, the second wave initiates its closing and the third wave completes the closing.

In GME_α and GME_β algorithms, the root of the tree is fixed. In the GME_α algorithm, the root, on initiating a session, informs all processes about it by broadcasting an open-session message via the tree (open session wave). On the other hand, in the GME_β algorithm, a process – starting from the root – sends information about opening of a session to only those children from which a request of the same type as the session has been received.

The root initiates closing of a session once it becomes aware of a pending request whose type is different from that of the session (that is, the request *conflicts* with the session). To close a session, a process – starting from the root – sends a close-session message to those children to which it had sent an open-session message earlier (close session wave). After sending close-session messages, a process waits until it has received a completion message from all those children and has also left its critical section, if applicable (completion wave). Once that happens, the process either sends a completion message to its parent (if a non-root process) or starts a new session (if the root process). Intuitively, sending of a completion message by a process signifies that the process and all its descendants have closed the current session and are possibly awaiting opening of a new session.

Intuitively, GME_α uses broadcast and convergecast to open and close a session, whereas GME_β uses *selective broadcast* and *selective convergecast* to open and close a session.

In the GME_γ algorithm, as opposed to the other two algorithms, root of the tree is not fixed and can change with time. Informally, once the current root of the tree has initiated closing a session and received completion messages for all its close-session messages, it selects one of the processes with an outstanding request to become the new root of the tree. The new root, which has a pending request itself, then initiates a new session whose type is same as that of its own request. This part of the GME_γ algorithm is similar to that of the GME_β algorithm.

Although in [21], GME_γ was presented as an extension of GME_β, GME_γ can also be viewed as an extension of the Raymond's tree based algorithm for mutual exclusion [5].

4. OUR ALGORITHM

Our algorithm is derived from the GME_γ algorithm proposed by Beauquier *et al*. Basically, we modify the GME_γ algorithm to further improve its performance as described next.

4.1 The Main Idea

The main idea behind our algorithm is to reduce the synchronization delay. The

synchronization delay in GME_γ consists of three parts: (1) a message chain from the current root of the tree to the last process to leave the critical section, (2) a message chain from that process to the current root of the tree, and (3) a message chain from the current root to the new root of the tree. The length of each message chain is bounded by $h_{\max}$ resulting in synchronization delay of at most $3h_{\max}$. To reduce the synchronization delay, we make two modifications to the GME_γ algorithm described next. When describing the two modifications, we use the phrase "*when the system is heavily loaded*" to mean that "*the queue of outstanding requests at the root is still non-empty immediately after the current session is initiated*".

First, when the system is heavily loaded, there is *no explicit* closing of a session. In Beauquier *et al.*'s algorithms [21], once a process learns about the opening of a session, it can enter and leave its critical section several times until it learns that the session has been closed or it generates a request that conflicts with the session. In our algorithm, once a process learns about the opening of a session, it can enter its critical section at most once. Intuitively, this modification causes the close session wave to be merged with the (corresponding) open session wave. We believe that such implicit closing of a session should not lead to much degradation in the performance when compared to the GME_γ algorithm, especially when the number of groups is large and all groups are equally likely to be requested. This is because, in that case, *the probability that two consecutive requests by a process belong to the same group is small*. With this modification, the worst-case synchronization delay reduces from $3h_{\max}$ to $2h_{\max}$.

Second, when the system is heavily loaded, the root of the tree for the next session is selected *at the same time* as the current session is opened. In the GME_γ algorithm, the root that initiates a session is also responsible for collecting completion messages from processes that execute their critical sections during that session (via selective convergecast). After collecting all required completion messages, it selects a new root, which is then responsible for opening the next session. In our algorithm, the root of the tree for the next session is responsible for collecting completion messages from processes that execute their critical sections during the current session. After the root for the next session has collected all completion messages for the current session, it initiates the opening of the next session. With this modification, the worst-case synchronization delay reduces from $2h_{\max}$ to only $h_{\max}$. We refer to our algorithm as OptimalGME.

Example 1: Consider three processes p, q and r at a distance of $h_{\max}$ from each other (see Fig. 1). Assume that p and q are currently executing their critical sections (which are of the same type), and p is the root of the current session. Further, r is the root of the next session. When GME_γ is used to manage critical section requests, before r can enter its critical section, in the worst-case, the synchronization delay may consist of a close session wave traveling from p to q, followed by a completion wave traveling from q to p, followed by transferring of root status from p to r. This results in synchronization delay of $3h_{\max}$. On the other hand, when OptimalGME is used to manage critical section requests, on leaving their critical sections, both p and q directly inform r, resulting in synchronization delay of only $h_{\max}$. ❑

Observe that, when the set of tree edges is fixed and messages are only exchanged over the tree edges, the lower bound on the worst-case synchronization delay (assuming

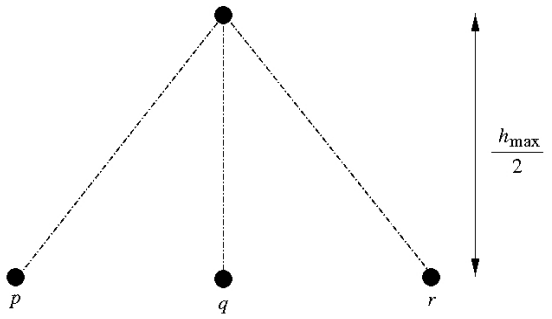


Fig. 1. An example to illustrate the difference between the synchronization delay of GME_γ and OptimalGME.

that the system is heavily loaded) for any group mutual exclusion algorithm is $h_{\max}$ – the diameter of the tree. Therefore our algorithm has optimal synchronization delay for its class of algorithms. Due to lack of space, a formal description of the algorithm and proof of the following theorems are provided elsewhere [22]. The first theorem states that our algorithm is correct.

Theorem 1 (correctness) OptimalGME satisfies safety, liveness and non-triviality properties.

The second theorem measures the performance of our algorithm in terms of message complexity, message-size complexity, waiting time and synchronization delay.

Theorem 2 (complexity) OptimalGME has message complexity of $3h_{\max}$, message-size complexity of $O(\min(m, n\log(m)))$, waiting time of $2h_{\max}$ and synchronization delay of $h_{\max}$.

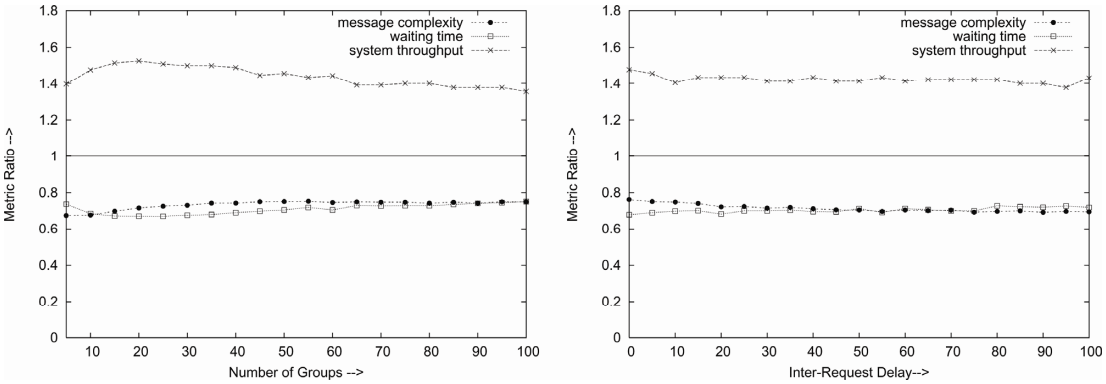
Note that OptimalGME has much better performance (lower message complexity, lower waiting time and lower synchronization delay) than GME_γ when the probability that two consecutive requests for critical section by the same process belong to the same group is small. On the other hand, if conflicts are rare, then GME_γ should have better performance than OptimalGME because a process can execute its critical section multiple times without exchanging any messages or incurring any (synchronization) delay.

4.2 Experimental Evaluation

We experimentally compare the performance of OptimalGME with Beauquier *et al.*'s third algorithm GME_γ using a discrete-event simulation. We compare the performance of the two algorithms with respect to three metrics, namely message complexity, waiting time and system throughput. To make it easier to compare the performance of the two algorithms, we report the ratio $\frac{\text{OptimalGME's performance}}{\text{GME}_\gamma\text{'s performance}}$ for each metric. Note that, for message complexity and waiting time, a ratio of less than one would imply that

OptimalGME has better performance than GME_γ. On the other hand, for system throughput, a ratio of greater than one would imply that OptimalGME has better performance than GME_γ.

Our experimental study has the following parameters. There are n processes requesting critical sections for m different groups. Processes are arranged in the form of a binary tree. (We have conducted experiments for other types of trees as well including a star and an inverted fork tree, and results are quite similar to those for a binary tree.) A process, on generating a request, randomly selects a group for the critical section. The inter-request delay (that is, duration of non-critical section) at each process is exponentially distributed with mean μ_{ncs}. Once a process enters a critical section, it departs after a delay that is uniformly distributed in the range [0, 2 * μ_{cs}] (duration of critical section). Message transmission delay (or channel delay) is modeled to follow an exponential distribution with mean μ_{cd}. In our experiments, parameters that have fixed values throughout are number of processes n, which is set to 25, and number of requests per process, which is set to 250. All other parameters are varied one by one to study their effect on the relative performance of the two algorithms.



(a) Varying the number of groups (m) with μ_{ncs} = 10 time units, μ_{cs} = 10 time units and μ_{cd} = 2 time units.

(b) Varying the mean inter-request delay (μ_{ncs}) with m = 50, μ_{cs} = 10 time units and μ_{cd} = 2 time units.

(c) Varying the mean time in critical section (μ_{cs}) with m = 50, μ_{ncs} = 10 time units and μ_{cd} = 2 time units.

(d) Varying the mean channel delay (μ_{cd}) with m = 50, μ_{ncs} = 10 time units and μ_{cs} = 10 time units.

Fig. 2. Relative performance of the two algorithms as a function of various parameters.

Fig. 2 depicts the variation in the ratios for the three metrics as a function of various parameters. The ratios are averaged over several runs to obtain 95% confidence level. As shown in the figure, OptimalGME outperforms GME_γ in all cases. Specifically, when compared to GME_γ, OptimalGME reduces the message complexity by as much as 35%, waiting time by as much as 40% and increases the system throughput by as much as 70%. The gap between the performances of the two algorithms is especially significant when the mean channel delay is large and the mean duration of critical section is small.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented an efficient algorithm for solving the group mutual exclusion problem for a tree network. Our algorithm has *optimal* synchronization delay when the set of tree edges is fixed and messages are only exchanged over the tree edges. An advantage of using a tree network is that group mutual exclusion algorithms developed for a tree network typically have lower message complexity than those for a fully connected network. Our simulation experiments show that, when compared to an existing algorithm for a tree network GME_γ, our algorithm can decrease the message complexity by as much as 35%, reduce the waiting time as much as 40%, and increase the system throughput by as much as 70%. Further, our algorithm retains all desirable features of GME_γ including bounded message sizes.

The tree in our algorithm (as well as Beauquier *et al.*'s algorithms [21]) is static in the sense that its set of edges remain fixed throughout the system execution. Only the relationship between neighboring processes (which process is parent and which is child) may change with time. As a result all messages generated by the group mutual exclusion algorithm are exchanged only over the tree edges even if the system contains other (non-tree) edges. For the traditional mutual exclusion problem, several tree-based algorithms have been developed in which the tree is *dynamic* instead of static (*e.g.*, [23]). With a dynamic tree, the overhead of exchanging messages is more evenly distributed over the communication network. To our knowledge, so far no tree based algorithm for group mutual exclusion has been developed in which the tree is dynamic in nature. As future work, we plan to develop such an algorithm for group mutual exclusion.

REFERENCES

1. E. W. Dijkstra, "Solution of a problem in concurrent programming control," *Communications of the ACM*, Vol. 8, 1965, pp. 569.
2. L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, Vol. 21, 1978, pp. 558-565.
3. G. Ricart and A. K. Agrawala, "An optimal algorithm for mutual exclusion in computer networks," *Communications of the ACM*, Vol. 24, 1981, pp. 9-17.
4. I. Suzuki and T. Kasami, "A distributed mutual exclusion algorithm," *ACM Transactionson Computer Systems*, Vol. 3, 1985, pp. 44-349.
5. K. Raymond, "A tree based algorithm for distributed mutual exclusion," *ACM Transactions on Computer Systems*, Vol. 7, 1989, pp. 61-77.
6. M. J. Fischer, N. A. Lynch, J. E. Burns, and A. Borodin, "Resource allocation with

immunity to limited process failure (preliminary report)," in *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, 1979, pp. 234-254.
7. E. W. Dijkstra, "Hierarchical ordering of sequential processes," *Acta Informatica*, Vol. 1, 1971, pp. 115-138.
8. K. M. Chandy and J. Misra, "The drinking philosophers problem," *ACM Transactions on Programming Languages and Systems*, Vol. 6, 1984, pp. 632-646.
9. K. M. Chandy and J. Misra, *Parallel Program Design: A Foundation*, Addison-Wesley, 1988.
10. Y. J. Joung, "Asynchronous group mutual exclusion," *Distributed Computing*, Vol. 13, 2000, pp. 189-206.
11. P. Keane and M. Moir, "A simple local-spin group mutual exclusion algorithm," in *Proceedings of the 18th Annual ACM Symposium on Principles of Distributed Computing*, 1999, pp. 23-32.
12. K. Alagarsamy and K. Vidyasankar, "Elegant solutions for group mutual exclusion problem," Technical Report, Department of Computer Science, Memorial University of Newfoundland, St. John's, Newfoundland, Canada, 1999.
13. V. Hadzilacos, "A note on group mutual exclusion," in *Proceedings of the 20th ACM Symposiumon Principles of Distributed Computing*, 2001, pp. 100-106.
14. Y. J. Joung, "The congenial talking philosophers problem in computer networks," *Distributed Computing*, 2002, pp. 155-175.
15. Y. J. Joung, "Quorum-based algorithms for group mutual exclusion," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 14, 2003, pp. 463-476.
16. M. Toyomura, S. Kamei, and H. Kakugawa, "A quorum-based distributed algorithm for group mutual exclusion," in *Proceedings of the 4th International Conference on Parallel and Distributed Computing, Applications and Technologies*, 2003, pp. 742-746.
17. R. Atreya and N. Mittal, "A dynamic group mutual exclusion algorithm using surrogate-quorums," in *Proceedings of the IEEE International Conference on Distributed Computing Systems*, 2005, pp. 251-260.
18. N. Mittal and P. K. Mohan, "An efficient distributed group mutual exclusion algorithm for non-uniform group access," in *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems*, 2005, pp. 367-372.
19. S. Cantarell, A. K. Datta, F. Petit, and V. Villain, "Token based group mutual exclusion for asynchronous rings," in *Proceedings of the IEEE International Conference on Distributed Computing Systems*, 2001, pp. 691-694,.
20. K. P. Wu and Y. J. Joung, "Asynchronous group mutual exclusion in ring networks," in *IEE Proceedings of Computers and Digital Techniques*, Vol. 147, 2000, pp. 1-8.
21. J. Beauquier, S. Cantarell, A. K. Datta, and F. Petit, "Group mutual exclusion in tree networks," *Journal of Information Science and Engineering*, Vol. 19, 2003, pp. 415-432.
22. V. Madenur and N. Mittal, "A delay-optimal group mutual exclusion algorithm for a tree network," Technical Report No. UTDCS-24-06, Department of Computer Science, the University of Texas at Dallas, U.S.A., 2006.
23. M. Naimi, M. Trehel, and A. Arnold, "A log(N) distributed mutual exclusion algorithm based on path reversal," *Journal of Parallel and Distributed Computing*, Vol. 34, 1996, pp. 1-13.

Vinay Madenur received his B.E. degree in Computer Science and Engineering from Madras University, Chennai, India in 2002 and M.S. degree in Computer Science from the University of Texas at Dallas in 2005. He is currently working as a Software Engineer at Qualcomm, Inc. in San Diego. His research interests include distributed systems, telecommunication network security and messaging services.

Neeraj Mittal received his B.Tech. degree in Computer Science and Engineering from the Indian Institute of Technology, Delhi in 1995 and M.S. and Ph.D. degrees in Computer Science from the University of Texas at Austin in 1997 and 2002, respectively. He is currently an assistant professor in the Department of Computer Science and a co-director of the Advanced Networking and Dependable Systems Laboratory (ANDES) at the University of Texas at Dallas. His research interests include distributed systems, mobile computing, networking and databases.