AN EFFİCİENT PERMİSSİON-BASED MUTUAL EXCLUSİON ALGORİTHM FOR MOBİLE AD-HOC NETWORKS

Sanjida Singhani[1], Poonam Saini[2]

[1]Assistant Professor, Dept of CSE, Chandigarh University, Gharuan, Punjab, (India)

[2]Assistant Professor, Dept of CSE, PEC University of Technology, Chandigarh, (India)

ABSTRACT

*The problem of Distributed Mutual Exclusion (DME) has been studied significantly over the years. The proposed protocols, in the existing literature, consider the various parameters, viz., liveness, fairness, message complexity and safety for optimization. The problem of DME can be handled using two approaches, namely, Token-based and Permission-based. In the paper, we propose an optimized Permission-based algorithm in which the total number of messages has been considerably reduced with the help of a new message "Hold". In parallel, we have also reduced the number of sites to which a node has to send "Hold" message by applying timestamp priority. Moreover, it will optimize t*n parameter, thereby, reducing the overall flow of "Hold" messages in the proposed protocol. The paper also discusses the correctness proofs for static analysis of the algorithm.*

Keywords: Critical Section, Distributed Mutual Exclusion; Message Complexity , Mobile Ad-Hoc Network.

I. INTRODUCTİON

A Mobile Ad hoc network (MANET) is a self configuring infrastructureless network of mobile devices connected by wireless channel [1]. It is an autonomous system of mobile nodes and associated hosts which collectively form an arbitrary and dynamic topology [2]. In distributed mobile ad-hoc networks, processes must share common hardware or software resources that assist each other to work independently at large scale [3]. Further, the access to a shared resource must be synchronized in order to ensure that, at any given time, only one process utilizes the available resources. Each process has a code segment called a critical section (CS) for accessing the shared resource [1]. Therefore, coordinating the execution of critical section is a big challenge. The problem can be handled by providing a finite-time *mutually exclusive* access to the CS. Also, each process must request permission to enter its critical section and release the same after exiting CS.

In the existing literature, there are two main approaches that have been proposed for solving the DME problem, namely, centralized and distributed [2, 3]. In centralized approach, one of the nodes acts as a central coordinator. Further, the central coordinator is fully responsible to store the complete information of incoming requests as well as available resources so that the shared resource is best utilized.

On the other hand, in distributed approach, the decision-making is distributed across the entire system. To accomplish the task of achieving DME using distributed approach, the two principles as follows:

a) The existence of token in the system leading to Token- based algorithms.

b) The collection of permission from nodes in the system leading to Permission-based algorithm.

A. Token-Based Approach

In token-based approach, there are two methods of using token for entering into CS. The first method states that only one process can enter CS by using a special object called token, which is unique to the whole system. Here, token acts as privilege to a process for entering the CS [2]. A process, the current owner of the token, selects the next token owner on the basis of priority. If no process wants to enter the CS, the token is held by the current process itself.

On the other side, in the second method, the processes are logically organized in a ring structure where the token is circulated from process to process, allowing them to enter into the CS [2]. After the process exits its CS, the token is released for further circulation. However, in case, the process is not interested to enter CS, it passes the token to the next node in the logical ring. Moreover, if the ring is unidirectional, starvation freedom is ensured.

Token-based approach has the following drawbacks:

- The token-based approach is highly prone to the loss of the token leading to a deadlock situation.
    - Existence of duplicate tokens causes problem.
- For uniqueness of token, complex token regeneration must be executed.

B. Permission-Based approach

In the permission-based approach, the process is allowed to enter CS by explicitly acquiring permission from a set of nodes or from all nodes, in the system. It is a non token-based approach. A priority in the form of logical clocks or timestamps can be established for incoming requests [3]. When a node completes its execution and exits from CS, it informs all other nodes from which it had received permission. Permission-based approach is further divided into two types, *i.e.*, *voting-based* and *coterie-based* [7]. In voting-based approach, vote assignment to each node, is done in the system itself. Therefore, a node that wants to enter CS, seeks permission from nodes that constitute the majority of votes. On the other side, in coterie-based algorithms, a collection of quorums (*i.e.*, set of nodes), is called a coterie which is attached to the system. Hence, a node seeking access to CS must obtain permission from each and every node of quorum in the coterie [7]. Figure 1 (given below) describes the flow of mutual exclusion algorithms.

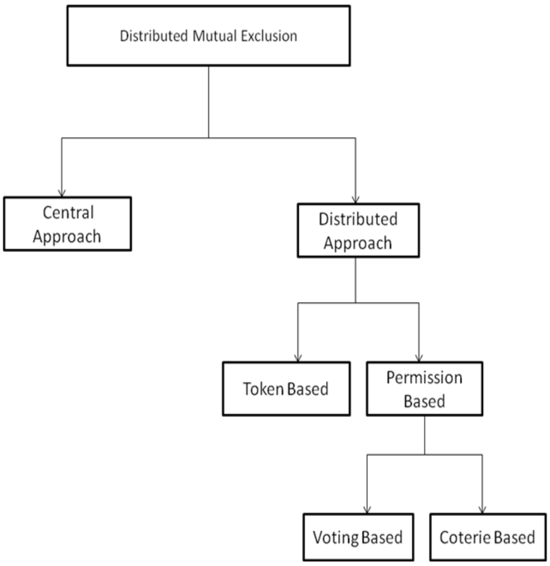

Fig 1: Flow Diagram of DME Approaches

II. RELATED WORK

A. Time, Clocks, and the Ordering of Events in a Distributed System **[Leslie Lamport, 1978].**

The first solution for distributed permission-based mutual exclusion problem was proposed by Lamport in 1978, popularly known as Lamport's algorithm. It uses three types of messages: request, reply and release. In order to serve the request messages, it uses the concept of logical clocks and assigns sequence numbers to the incoming request i.e., timestamp [4]. Thereafter, every node maintains a queue of pending requests for entering into the CS. When a node n_i wants to execute CS, it broadcasts message to all other nodes and its corresponding request is stored in a local queue. Further, a node n_j after receiving the request message from n_i, stores the message in its own queue and sends a timestamped reply message. However, a node n_i can access CS only when two conditions are satisfied: Firstly, it must have received reply messages from all other processes with timestamps greater than its own timestamped request. Secondly, the process' own request must be at the front of its queue. When n_i exits from the CS, it broadcasts a release message. The message complexity of this algorithm is 3(N-1).

B. An Optimal Algorithm for Mutual Exclusion in Computer Networks **[G. Ricart and Ashok K. Agrawala 1981].**

Ricart and Agrawala (RA) improved Lamport's solution by reducing message complexity from 3(N-1) to 2(N-1). The algorithm uses request and reply messages only, thereby, avoiding the release messages [5]. In case, each node, either in CS or requesting CS, has a higher priority request, would not send the reply messages. A node enters CS only after receiving permission from all nodes. When it exits from CS, it sends all reply messages that have been deferred so far.

C. Maekawa's O(√n) Distributed Mutual Exclusion Algorithm **[M. Maekawa 1985].**

Maekawa's algorithm introduced the concept of coterie by associating each node with a set of nodes. In its design, there is always a node in the intersection of two subsets [7]. A node n_i must obtain permission from all other nodes in its home set, S_i, before it can enter its CS. After exiting CS, it replies to node at the top of requesting queue instead of sending message to all nodes in the queue. The number of messages required to handle a request is 3 times the size of the request set [7]. For a system with N nodes, the size of each request set is roughly square root of N, therefore, total message complexity is $3\sqrt{N}$.

D. *Distributed Mutual Exclusion Algorithm for Mobile Computing Environments* **[Mukesh Singhal and D. Manivannan Singhal 1997].**

The authors proposed a concept of “look-ahead” technique to handle DME for infrastructured networks in mobile environment. The technique, instead of enforcing mutual exclusion among all the sites of a mobile system, enforces it only among the sites concurrently competing for CS [9]. This results in reduced message overhead. Further, “look-ahead” mutual exclusion algorithms eliminates unnecessary communication among sites, hence are more efficient. Here, message complexity is proportional to average number of active sites at any time instead of the total number of sites in the system.

E. *A Scalable Mutual Exclusion Algorithm for Mobile Ad Hoc Networks* **[Weigang Wu, Jiannong Cao, Jin Yang in 2005].**

The paper presented the first permission-based solution for DME problem for MANETs. However, it uses "look-ahead" technique, presented by M. Singhal [10] (for infrastructured mobile networks). The proposed protocol has reduced message complexity in MANET environment. Also, the authors presented timeout mechanism to deal with MANET susceptibility to link and host failures. It provides better performance under high load situations, *i.e.*, when more mobile hosts are active. Furthermore, the paper describes a conventional method for fault tolerance in MANETS [9].

F. *A Reliable Optimization on Distributed Mutual Exclusion Algorithm* **[Moharram Challenger, Peyman Bayat and M.R. Meybodi 2006].**

The paper proposes an asynchronous message passing algorithm for distributed system. In their work, notable improvements are made on the number of messages exchanged. Like, a process P_i on finishing CS sends a FLUSH message to the concurrently requesting process, along with the next highest priority request, whose requests was earlier deferred [11]. By examining these requests, P_i can determine the order in which these processes will execute CS. Using this, the following optimization are made. Assume P_k has the highest priority among all request messages. Then, P_i can send reply just to P_k, apprising P_k of all the information that P_i has gathered instead of replying to m nodes. This leads to reduced message complexity [11], hence, enhancing the performance of the system. Its message complexity fluctuates between (N-1) and 2(N-1) per critical section access.

G. *A Novel Permission-based Reliable Distributed Mutual Exclusion Algorithm for MANETs* **[Murali Parameswaran and Chittaranjan Hota in 2010].**

The approach introduced a new message called *“Hold”* in order to ensure that the requesting nodes are aware of the currently executing node in CS. It uses an adaptable timeout mechanism to deal with critical sections having varying execution times [13]. The paper presents an algorithm that can handle situations where the node in critical section itself can fail, with the help of the *“Hold”* message and the adaptive timeout mechanism. It also informs about the expected time a node remains in CS. Thus, it also resolves the issue that if a node has crashed or executing a lengthy process. The major drawback is the increased message complexity of the algorithm with the introduction of new message *“Hold”*. The improvement could be the reduction in the number of the sites to which *“Hold”* has to be sent.

III. INFERENCES AND MOTIVATION

From the review of existing literature, the following inferences have been drawn:

1. Lamport's algorithm suffered high message overhead. Moreover, the algorithm does not handle failures to make the system fault-tolerant.
2. Ricart Agrawala proposed an improved version of Lamport's algorithm. However, it suffered from single point of failure as well as incurs high message complexity.
3. In Maekawa algorithm, there is no defined order for messages that are sent to the subset of nodes, which in case of communication delay, leads to deadlock situations.
4. Communication delays are typical in a MANET environment. To handle this, new algorithms were proposed with new message like *FLUSH* and *Hold.* Although, the protocols resolves deadlock problem, however, they incurs increased message complexity.

The prime motivation of our proposed algorithm is to ensure that there is less message traffic with the existence of new message and at the same time guarantees deadlock freedom. The proposed approach works on the reduction of number of sites to which a node has to send "*Hold*" message by applying timestamp priority. This will optimize the reduced flow of messages in the system.

IV. SYSTEM MODEL AND ASSUMPTIONS

We consider a MANET comprising of N nodes (N_0-$N_{(n-1)}$), each having unique identification number, *Idno* and a particular timestamp value, T_cs_i (timestamp value of i^{th} node to retain the critical section). Further, the mobile nodes forming dynamic topology communicate with each other as well as access the shared resource in a wireless channel through asynchronous message exchanges. Moreover, only one process accesses the available shared resource. Therefore, the requesting nodes are notified about the exit time of the current node, to access the critical section. This will also ensure that the current node in the CS has not arbitrarily failed or crashed. It has been presumed that Link and Node failures may occur, although, data can be recovered, either by resetting the values, or by using older set of values. The system model imposes a finite time on the access of CS by a particular node, thereby, maintain liveness into the system.

V. OVERVIEW OF ALGORITHM

A. Data Structures Used

i) ***Idno***: A unique identification number of each node.

ii) ***REQ_que***: A queue which is maintained by the node in the CS to keep track of the Request messages to access CS.

iii) ***HOLD_que***: A queue which is maintained by the node, currently in the CS, to keep the track of number of nodes to send the "Hold" messages.

iv) ***T_req***: A vector to keep track of the timeout values of REQ messages.

v) $\boldsymbol{T\_cs_i}$**:** A vector to keep track of time upto which a node retains the CS.

vi) ***Tcs_exit***: A vector to maintain the amount of time left for current node to exit CS.

vii) **Inft_set:** An array maintained by each node to keep track of nodes to send REQ message and seek permission before entering CS.

B. Types of Messages Used

- *Request for Critical section, REQ*: When a mobile node wants to access critical section, it will send request, REQ, to all nodes in its *Inft_set*. However, the nodes which are not demanding CS, will respond to the requesting node by sending immediate Reply or "*Hold*" message. Also, *T_req* is set, which gives the estimate of round trip time between nodes.
- *Reply message*: When the nodes in the *Inft_set* gets REQ, which contains identification number and timestamp value. Nodes themselves check if they are requesting for CS or not, then they send immediate Reply message (if not requesting). After getting Reply from all nodes in its *Inft_set*, it enters CS.
- *Hold message*: While the node is in the critical section, if it gets REQ then it will send "*Hold*" message which encloses *Tcs_exit*, which specifies the amount of time left for it to exit the critical section.

In MANETs, suppose, there are many nodes that are requesting for CS simultaneously, sending "*Hold*" message to all by current node in CS becomes overhead. Therefore we use the concept of low timestamp value here, the nodes with low timestamp value in their REQ will be send "*Hold*" message to notify the amount of time left by current node to exit CS.

C. Brief Outline

In the mobile ad hoc environment, the working of proposed algorithm is divided into two scenarios. In both the scenarios, the commonalities are:

i) We assume that there are four mobile nodes, N_0,N_1,N_2 and N_3 forming MANET.

ii) Each mobile node has its own identification number, *Idno.*

iii) When a node wants to enter CS, it sends request to other nodes and waits for their Reply, thereby using them as permission (either "*Hold*" or Reply) to enter into the CS.

D. Scenario1

Initially, we assume that there is no node in the CS and also, Request queue, *REQ_que* is empty. Suppose, at some interval, mobile node N_0 wants to enter CS. It sends Request, REQ that possess its identification number and timestamp value, to its own *Inft_set*. All nodes in the *Inft_set*, if not interested in accessing CS, will reply to the node N_0 by sending Reply message as permission to the node. After obtaining all Replies, N_0 enters into CS. The algorithm for entering into CS is discussed below in the form of pseudo code given in table 1:

```
//mobile node $N_0$ wants to enter CS

proc Send_REQ
Begin
{
    Set $N_0$ $Idno$;
    Set $N_0$ $T\_cs_i$ ;          //time to retain CS
    for ($N_1$, $N_{2,}$ … $N_n$ ε $info\_set_0$)
{
      Send REQ ($Idno$ + $T\_cs_i$ );
      Set $T\_req$ for every REQ ;
        If ($N_1$, $N_2$ ... $N_n$ are not demanding CS)
        {
          Send "Reply" to $N_0$;
        }
        $N_0$ enters CS.
 }
 }
End
```

Table1. Algorithm for requesting CS

E. Scenario2

In the second scenario, we have proposed an algorithm where N_0 is already in CS. Further, N_2 and N_3 want to access the CS simultaneously. Nodes N_2 and N_3 will send REQ embedded with the timestamps, to their corresponding *Inft_sets*. As N_0 is present in the *Inft_set* of both the nodes, timestamp priority is used to break the symmetry of concurrent request messages. Among the requesting nodes, the one with low timestamp, *T_cs* , will receive "Hold" message from N_0 , shown in Fig2

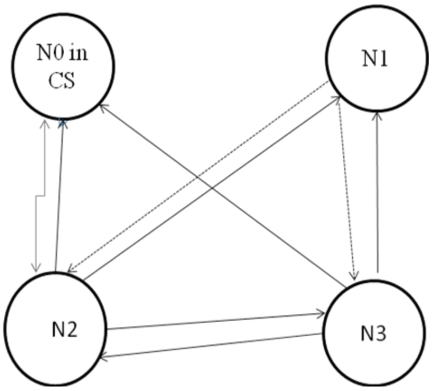

Fig 2: Concurrent request from N_2 and N_3 while N_0 in CS

Request message	⟶
Reply message	⇢
Hold message	⇆

Following table presents the pseudo code of the second scenario:

```
// $N_0$ in CS, $N_2$ and $N_3$ demands for CS simultaneously
Begin
     Set $N_2$ Idno + $T\_cs_i$;
     Set $N_3$ Idno + $T\_cs_i$
     $N_2$ send REQ ( $N_0$, $N_1$, $N_3$ ε info_set$_2$ );
    $N_3$ send REQ ( $N_0$, $N_1$, $N_2$ ε info_set$_3$ );
 If ( $N_1$ doesn't demand CS )
 {
 Send  "Reply message" ;
 }
 else  $N_2$ and  $N_3$  waits;
 for ( $N_0$  in CS )
{
 Add  $N_2$ and  $N_3$ REQs to REQ_que of  $N_0$ ;
 Compare $T\_cs_i$ of all REQ (REQ_que);
 Send "Hold" message to low $T\_cs_i$,  $N_2$  ;
 Set  Tcs_exit;    // for every "Hold" message//
Add  $N_3$  to "HOLD"_que;
}
                    $N_0$ exit CS;
                      $N_2$  enters CS;
}
End
```

Table 2: Algorithm for Hold Message

VI. PROOF OF CORRECTNESS

The section discusses the proof of three properties *Liveness, Fairness and Safety*, to ensure the correct working of the proposed algorithm.

Theorem 1: With the help of "Reply" and "Hold" message, the algorithm ensures fairness as well as determines the waiting time.

Argument: Assume that a mobile node N_j wants to access critical section while another node N_i is already executing CS. Any site that belongs to the information set as well as requesting CS simultaneously, receives either a Reply message or a "*Hold*" message. The Reply messages are sent immediately by the nodes which are not demanding CS access. On the other side, the "*Hold*" message is sent by the node N_i, currently in CS. This informs about the waiting time to the requesting node. Further, if more than one process request for CS at the same time, the decision of sending "*Hold*" message is made on the basis of their timestamp values. Also, it proves that only one process per node executes the CS

Theorem 2: The algorithm ensures liveness.

Proof: Presuming a situation, when more than one node requests for the CS access, simultaneously. Since, each request in the proposed algorithm is timestamped, which is already received by every node in the *Inft_set*. Therefore, based on the timestamp priority, requesting node with lower timestamp value will be sent a *"Hold"* message. Also, the node is notified about its waiting time. This ensures CS availability to all nodes and therefore, guarantees liveness of the system.

Theorem 3: The algorithm ensures Safety.

Proof: Without the loss of generality, frequent node/link failures occur in dynamic MANET environment. This results in the loss of messages. If the failed or crashed link/node is not in *Inft_set* and is not waiting for Reply, then there will be no effect on the execution. Whenever link/node failure occurs, it will recover after retrying time period or resetting to the older values. Thereafter, resuming to its normal functions, it can participate in the network execution, thereby, ensuring safety.

VII. PERFORMANCE

In proposed algorithm, the message complexity will exceed $2(\varphi-1)$ because of using additional "*Hold*" message. However, the proposed algorithm reduces total number of "Hold" messages when compared to [13]. The message complexity will be $[2(\varphi-1) + t*n]$ where t is the timeout period and n is number of nodes in HOLD_que. In the algorithm, we have optimized the $t*n$ parameter by reducing n factor and applying timestamp priority, thereby, leading to controlled flow of messages in the system.

VII. CONCLUSION

Various solutions have been proposed in the literature for achieving DME using *Token-based* and *Permission-based* approach. In Permission-based solutions, a process that requests to access CS must receive permission from all nodes in its information set by message exchanges. However, the number of messages exchanges is large in the existing literature. The proposed work focus on the reducing the number of message exchanged including new message *"Hold"*, thereby, optimizing $t*n$ parameter. Further, this reduces the overall latency, thus, increasing the performance of the system.

REFERENCES

[1] B.D. Kshemkalyani and M. Singhal, Distributed mutual exclusion algorithms in Distributed Computing Principles, Algorithms and Systems, 1st ed. Cambridge University Press, May 2008.

[2] G. Coulouris, J.Dollimore, Tim Kindberg, Distributed System concept and Design Addison-Wesley, Pearson Education, 2001.

[3] Tanenbaum, A.S., and Steen M.V, Distributed Systems Principles and Paradigms, Prentice-Hall International, Inc, 2002.

[4] Lamport, L. “Time, clocks and the ordering of events in a distributed system.,” Comm. A CM 21, pp 558-565, 7 July 1978.

[5] G.Ricart and A. K. Agrawala, “An Optimal Algorithm for Mutual Exclusion in Computer Networks,” Communications of the ACM, pp 9-11, 1981.

[6] Maekawa, M., Oldehoeft, A.E., and Oldehoeft, R.R, Operating Systems Advanced Concepts, Menlo Park, CA: Benjamin/Cumings, pp:200-208, 1978.

[7] M Maekawa, “A √N algorithm for mutual exclusion in decentralized systems,” ACM Trans on Computer Systems, Vol. 3, No 2, pp. 145-159, May 1985.

[8] M. Singhal, “A Taxonomy of Distributed Mutual Exclusion”, Journal of Parallel and Distributed Computing 18(1), pp.94-101, 1993.

[9] M. Singhal, and D. Manivannan, “A Distributed Mutual Exclusion for Mobile Environments”, Proc. IASTED Intl. Conf. on Intelligent Systems, pp 557-561, 1997.

[10] Weigang Wu, Jiannong Cao, Jin Yang, “A Scalable Mutual Exclusion Algorithm for Mobile Ad Hoc Networks,” Proc. of the 14th International Conference on Computer Communications and Networks (ICCCN2005), San Diego, USA, Oct. 17-19, 2005.

[11] Moharram Challenger, Peyman Bayat and M.R. Meybodi, “A Reliable Optimization on Distributed Mutual Exclusion Algorithm” TRIDENTCOM, 2006

[12] Bharath Kumar A.R. and Pradhan Bagur Umesh , “An Improved Algorithm for Distributed Mutual Exclusion by Restricted Message Exchange in Voting Districts” in 11th International Conference on Information Technology, 2008.

[13] Parameswaran, Murali; Hota, Chittaranjan, “A novel permission-based reliable distributed mutual exclusion algorithm for MANETs,” Wireless And Optical Communications Networks (WOCN), 2010 Seventh International Conference On, vol., no., pp.1-6, 6-8 Sept. 2010.

[14] Parameswaran, M.; Hota, C., “Arbitration-based Reliable Distributed Mutual Exclusion for Mobile Ad-hoc Networks”, Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks, 2013 11th International Symposium and Workshops on, vol., no., pp.380-387, May 13-17, 2013.

[15] K. Erciyes, “Distributed mutual exclusion algorithms on a ring of clusters” ,Proc. of International Conference on Computational Science and Its Applications ICCSA 2004, vol. 3045/2004, LNCS, SpringerVerlag, May 2004, pp 518–527. doi: 10.1007/b98053.

[16] I. Suzuki and T. Kazami, “A distributed mutual exclusion algorithm,” ACM Trans on Computer Systems, Vol.3, No.4, pp 344-349, Nov 1985.