***I.J. Wireless and Microwave Technologies*, 2015, 4, 1-14**
Published Online July 2015 in MECS(http://www.mecs-press.net)
DOI: 10.5815/ijwmt.2015.04.01

***Available online at* http://www.mecs-press.net/ijwmt**

Dynamic Request Set based Mutual Exclusion Algorithm in MANETs

Ashish Khanna[a,*], Awadhesh Kumar Singh[b], Abhishek Swaroop[c]

[a]*MAIT, Delhi-110085, India*
[b]*NIT, Kurukshetra-136119,Haryana, India*
[c]*Galgotias University, Greater Noida-201306, U.P., India*

Abstract

Mutual Exclusion (*ME*) problem involves a group of processes, each of which intermittently requires access to the only resource present in the environment. Handling *ME* problem becomes difficult due to the dynamic nature of the ad hoc environment. This paper presents a token-based solution to *ME* problem in the mobile ad hoc environment. The proposed token based algorithm is sensitive to link forming and link breaking and thus is suitable for *MANET*. The algorithm uses the concept of dynamic request set (*DRS*). As the request set is dynamically updated, the average size of request set is reduced resulting in less number of messages exchanged per critical section. The algorithm satisfies mutual exclusion, starvation freedom, and freedom from deadlock. The present algorithm has been compared with *DRS* based *ME* algorithms for static distributed systems. The results show that the concept of *DRS* in *MANETs* can be successfully used. Token loss problem has also been handled separately in the present exposition.

Index Terms: Resource Allocation, Mobile Ad Hoc Networks (MANETS), Distributed Algorithms, Distributed Computing, Mutual Exclusion.

1. Introduction

Mobile ad hoc network (*MANET*) contains mobile hosts which move arbitrarily in the environment. Communication topology may change with time as the hosts move into and go out of each other's transmission range. Similar to static networks, the efficient and correct use of shared resource(s) is one of the important problems in *MANETs*. Over the years, researchers have proposed several resource allocation problems in distributed systems e.g., mutual exclusion [1], dining philosopher problem [2], and drinking philosopher problem [3].

* Corresponding author. Tel.: +911127582080
E-mail address: ashishk746@yahoo.com

The mutual exclusion [1] problem is one of the most fundamental problems in distributed system. In Mutual exclusion (*ME*) problem, a several processes intermittently require entering the Critical Section (*CS*) in order to gain exclusive access to the shared resource. A solution to the ME problem must satisfy the following three correctness properties [4]:

- Mutual Exclusion (safety): At most one process is allowed to enter the *CS* at any moment.
- Deadlock Freedom (liveness): If any process is waiting for the *CS* then in a finite time some process enters the *CS*.
- Starvation Freedom (fairness): If a process is waiting for the *CS* then in a finite time the process enters the *CS*.

The solution to the mutual exclusion problem can be categorized into two classes: *token-based* and *permission-based* [1, 5]. In token-based algorithms, a unique token is shared among the nodes. A node is allowed to enter the *CS* only if it possesses the token. While in a permission-based algorithm, the node that wants to enter the *CS* must first obtain the permissions from other nodes by exchanging messages.

In MANETs, the nodes need to share resources; hence, an efficient solution of mutual exclusion problem in *MANETs* is essentially required. However, due to the mobility of nodes, the solutions developed for solving mutual exclusion problem in static networks need to be adopted and modified before these can be effectively used in *MANETs*.

The token-based algorithms can be classified as static request set based [6] and dynamic request set (*DRS*) based algorithms [7, 8]. In static request set based algorithms, a node requesting for token always select the same set of nodes to which the request has to be sent. On the other hand, in *DRS* based algorithm, the set of nodes to which a requesting node need to forward its request for token changes dynamically. Generally, the number of messages per critical section is less in *DRS* based algorithms in comparison to static request set based algorithms. Hence, in this paper, a token based solution for mutual exclusion problem in *MANET's* using the concept of dynamic request set [7], has been proposed.

The rest of the paper has been organized as follows. Section 2 discusses the related work. In section 3, system model, algorithm concept and data structure have been presented. The proof of correctness and the performance analysis has been shown in section 4 and section 5 respectively. Section 6 presents simulation results whereas section 7 handles token loss. Conclusions and future scope is shown in section 8.

2. Related Work

In 1993, Badrinath-Acharya-Imielinski [9] proposed two distributed mutual exclusion algorithms for cellular wireless networks. Both are adaptations for cellular networks, these adaptations avoid communicating frequently with hosts and finally reduce considerably the cost (for circulating token). Later on, *MANETs* emerged as an important subclass of wireless networks. Walter-Kini [10] proposed the first algorithm for solving the mutual exclusion problem in *MANETs* using a directed acyclic graph (*DAG*). The algorithm is suitable for mobile environment since nodes are required to keep information only about their immediate neighbours. Walter-Welch-Vaidya [11] proposed a reverse link (*RL*) based algorithm in which the requests are forwarded to the token holder over the tree and the token is delivered over the reverse tree path to the requesting node. The token holder will always be the lowest node in the *DAG*. The algorithm proposed by Roberto-Baldoni-Virgillito [12] is based on a dynamic logical ring and combines the two methods token-asking and circulating token. The algorithm aims to maintain device power consumption as low as possible by reducing the number of hops traversed per *CS* execution and by avoiding sending any control message when no process requests the *CS*.

Another variant of the classical mutual exclusion problem is *local mutual exclusion*. In local mutual exclusion, the nearby nodes (i.e. nodes in the same neighbourhood) compete with each other for the exclusive access of a shared resource. The local mutual exclusion problem has been discussed earlier in static networks [2, 13, 14]. Attaiyya et al. [15, 16] defined the local mutual exclusion problem in *MANETs*. They proposed two algorithms using doorways for solving local mutual exclusion problem in *MANETs*. Khanna-Singh-Swaroop [17] defined *k-local mutual exclusion problem* (*KLME*) in *MANET's* and proposed a token-based solution for the same. Khanna-Singh-Swaroop [18, 19] proposed a variant of the group mutual exclusion problem and named it as *Group local mutual exclusion problem [GLME]*. Wu-Cao-Raynal [20] defined a new problem named *LGME* (Local group mutual exclusion) as a variant of group mutual exclusion problem especially suited for *VANET's* and proposed a coterie based solution for *LGME*.

In recent years, several researchers focused on solving mutual exclusion problem and its variants in *MANETs*, therefore several algorithms to solve ME problem in MANETs [21, 22, 23, 24, 25, 26, 27, 28] have been proposed. Whereas, Chang-Singhal-Liu [7] used the concept of dynamic request set in order to reduce the message complexity of Suzuki-Kasmi algorithm [6] which was based on static request set. The simulation results presented in [7] shows that the message complexity can be reduced up to 60% using *DRS* in comparison to static request set. This motivated us to develop a token-based algorithm based on *DRS* to solve the mutual exclusion problem in *MANETs*.

3. System Model

In the present *MANET* environment there is *n* number of mobile nodes. The mobile nodes compete for the resource available in the region.

In the present paper, following assumptions have been made:

1. Number of nodes is finite and each node has unique identifier.
2. Any node can join or leave the ad hoc network.
3. There is no message loss.
4. Message propagation delay is finite but unpredictable.
5. The maximum time for which a process can be in its *CS* is bounded.
6. A process doesn't make a new request till its old request is satisfied.
7. Asynchronous distributed environment for the ad hoc network has been assumed.
8. Communication between the nodes is through message passing.
9. A workable link layer protocol is in place.
10. Underlying ad hoc network is of quasi stable nature.

The similar assumptions have been made by previous works on *MANETs* [11].

3.1. Working of Algorithm

For the convenience of the reader, a high level working of the proposed algorithm has been presented below. However, the detailed pseudo code and state transition diagram has also been presented in this section.

Initially, all other nodes are included in the request set of each node except *node 1* which is assumed to have token initially and whose request set is set to be empty. If a requesting node has the idle token it enters *CS* immediately. Otherwise, it will send request to all nodes in its request set and wait for the token. On receiving a non-stale request, the corresponding request number is updated in the request array of the receiving node. Now, based upon the state of the node, there are four possible cases. (i) The token is in idle state. The token is forwarded to the requesting node and requester's id is added in the request set of the token forwarding node. (ii) The token holding node is in *CS* state. The appropriate action will be taken when the node will come out of *CS*. (iii) The node is in requesting state. The node checks if the requesting node is in its request set. If not, the receiving node forwards its own request to the requester and adds requesting node's id in its request set. (iv) The node is in remainder state. The requesting node's id is added in the request set of receiving node, if it is not already there.

After receiving token, the new node will set its request set as empty. If it is in requesting state it enters *CS*, otherwise it will hold the idle token. After exiting *CS*, the node updates the token queue. In case token queue is not empty, it adds all nodes in the token queue in its request set and transfers the token to the node at the front of token queue. Otherwise, the node will hold the idle token.

An outgoing node checks whether it is a token holder node or not. In case it is a token holder node and its token queue is not empty it will forward the token to the node at the front of token queue, however, if token queue is empty it will send the token to the lowest id node. The information that the node is leaving the ad hoc region is broadcasted along with the token information.

After a node receives leaving information from any node, it will update the token information only if it receives the same and deletes leaving nodes information from its local data structure. In case, the token holder receives the leaving information it deletes the leaving nodes request if present in the token queue.

In case, a new node joins the ad hoc region its local data structure is initialized and its joining information is broadcasted. A node receiving the joining information of any node updates joining node's information in its local data structure.

Fig. 1 shows the state transition diagram of the *DRS* algorithm in MANETs. This diagram shows four states of a node and change of state when any event is performed on that node. The four states are *HI*: holding idle token, *REM*: Remainder, *CS*: Critical Section and *WAIT*: Waiting.

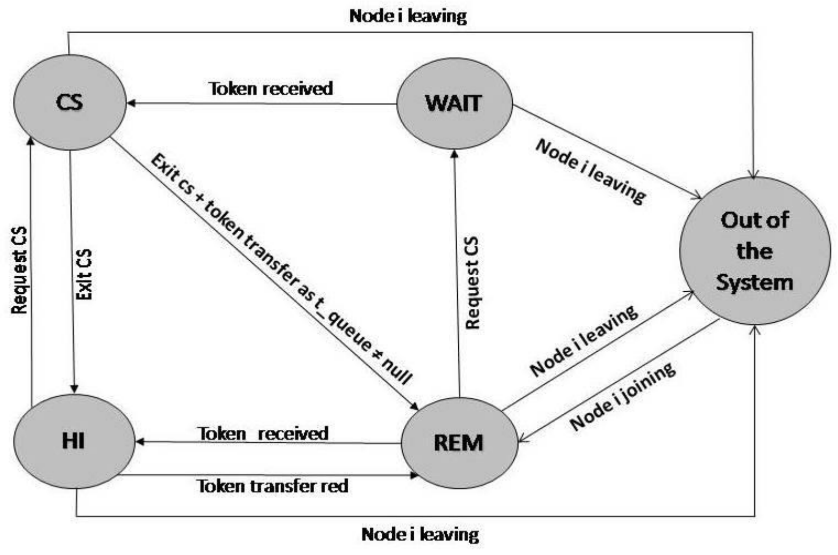


Fig.1. State Diagram of node i

3.2. *Data Structure*

Data Structure at node *i*.

a) n: Number of nodes inside the ad hoc environment.
b) $RN_i[j]$: Node i request number list where $1<j<N$
c) RS_i: Request Set of Node *i*
d) $leader\_node_i$: Stores leader information
e) $state_i$: State of Node *i*
   REQ: Requesting

REM: Remainder
CS: Critical Section
HI: Holding idle token

f) Have_Token$_i$: Boolean variable indicating whether node has token or not

Data structure at Token.

a) LN: Latest request number list
b) T_queue: Token Queue

Messages:

a) Request_CS(*i*, RN): Request message with respective request number.
b) token(T_queue, LN): A node may enter *CS* only if it has received token.
c) i_am_leaving(RN$_i$, RS$_i$, X): Broadcasted message when node *i* is about to leave the ad hoc environment with corresponding request number and token information (if any).
d) I_have_join(*i*): Broadcasted message when node *i* joins the ad hoc environment.

3.3. Event Driven Description of the Algorithm at node i

a. Initialization

```
for(i = 0; i < n; i++)
   for(j = 0; j < n ; j++)
     RN_i[j] = 0; RS_i = all nodes inside environment except i
     state_i = REM; Have_Token_i = FALSE; leader_node_i = -1
// initialization at token holder
Let node i is holding the token
Have_Token_i = TRUE; RS_i = Ø; T_queue = Ø; leader_node_i = i; state_i = HI
for(j=0; j < n; j++)
     LN[i] = 0
```

b. Request_CS: node i request for CS

```
state_i = REQ
if(Have_Token_i ! = i)
   RN_i[i] = RN_i[i]+1; Send Request_CS to all nodes in RS_i
else
   state_i = CS; enter CS; call exit_CS
```

c. Received Request_CS(j, p): receiving of request for CS by node i from node j

```
if (p > RN_i[j])
  RN_i[j] = p
   if (Have_Token_i  &&  state_i = HI)
      Have_Token_i = FALSE; set state_i = REM; Send token to node  j
      if(RS_i ∉ j)
         RS_i= RS_i ∪ {j}
   elseif (state_i = REQ && RS_i ∉ j)
      send Request_CS to j; RS_i = RS_i ∪ {j}
   elseif (state_i = REM)
      if (RS_i ∉ j) RS_i=RS_i ∪ {j}
```

d. Rec_token(T_queue, LN): token received by node i

```
Delete the entry of node i if present in token queue
Update $RN_i$; $RS_i = Ø$; $Have\_Token_i$ = true
if($state_i$ = REQ)
   $state_i$ = CS; Enter CS; Exit_CS
else
   $state_i$ = HI
```

e. Exit_CS: node i exited from CS

```
LN[i] = $RN_i[i]$
for(p = 1; p < n; p++)
    if($RN_i[p]$ = LN[i]+1 && p $\notin$ T_queue)
      add entry of p in token queue
add all elements of token queue in $RS_i$
if(T_queue != NULL)
   $Have\_Token_i$ = FALSE; send token to the node at front of token queue
else
   $state_i$ = HI
```

f. Node_leaving: node i is about to leave the ad hoc environment

```
if($Have\_Token_i$ = TRUE)
   if(T_queue != NULL)
      send token to X top element of T_queue; $RS_i = RS_i \cup \{X\}$
   else
      send token to lowest id X; $RS_i = RS_i \cup \{X\}$;
      $Have\_Token_i$ = FALSE
else X = -1
   broadcast i_am_leaving($RN_i$, $RS_i$, X) to all nodes in the ad hoc region
```

g. Rec_i_am_leaving: Node i receives I_am_leaving message from node j

```
Update new token holder information and Request Set information if it has been received; Delete information
of j from local data structure
 if($Have\_Token_i$ = TRUE && j $\in$ T_queue)
    Delete j from token queue
```

h. New node i joins the ad hoc environment

```
Initialization of local parameters; $RS_i$ = all node in the ad hoc environment except itself
Broadcast I_have_join
```

i. Node i receives I_have_join(j)

```
Update entry of node i in the local data structure of node i
```

4. Proof of Correctness

4.1. Mutual Exclusion

Theorem 1. At any point of time, there can be at most one node inside *CS*.

It has been assumed in the system model that no message is lost during transit and token is generated at the time of initialization. In initialization, only one token is generated, hence, only a single node can have token and the corresponding node holding the token can enter *CS*. Moreover, the outgoing token holder node transfers the token to the node at the front of the token queue or to the lowest id node (if token queue is empty). Hence, at any point of time only one node may hold the token and *theorem 1* is proved.

4.2. Freedom from Starvation

Lemma 1. $(\forall P,Q)\ P \in RS_Q$ *or* $Q \in RS_p$ is always satisfied.

The algorithm initializes the request set and maintains the request set in the manner similar to Chang-Singhal-Liu [7]. In [7], It has been proved that: $(\forall P, Q)\ P \in RS_Q$ *or* $Q \in RS_p$ is always satisfied in case of static distributed systems. In ad hoc environment a node may go out and can join the network. An outgoing node may be token holder node or non token holder node. A token holder node transfers the token either to the node at the front of the token queue or to the node with lowest id (in case token queue is empty). This information is broadcasted to all nodes and the request sets are updated accordingly ensuring the required condition. Moreover, when a new node joins the ad hoc region, it is not in the request set of any other node, however, it contains all other nodes in its request set (Initialization process). Therefore, the condition still remains satisfied. It has been shown that the $(\forall P, Q)\ P \in RS_Q$ *or* $Q \in RS_p$ remains true all the time. Hence, the *lemma 1* is proved.

Lemma 2. The request of a requesting node eventually reaches the token holder node.

Let *P* is a requesting node and *w* is the token holder node. Now there are two possibilities a) $P \in RS_w$ b) $w \in RS_p$ (from *lemma 1*)

a) *w* must have sent a request to *P*, after receiving this request *P* will add *w* in its request set and forward its request to *w* which will be received by the *w* and is added in token queue.

b) Since, $w \in RS_p$ than *P* must have sent a request to *w* hence its request will be added in the token queue.

In all possible cases the request is added in token queue and *lemma 2* is proved.

Theorem 2. Every request will eventually be served.

The request of every requesting node will eventually be added in token queue (from *lemma 2*). Now, the token queue is *FIFO*, the token is transferred always to the node at the front of token queue and all the additions are done at the rear of the token queue. Moreover, the time spent by a process in the *CS* is finite. Hence, any request added in the token queue will ultimately reach at the front of the token queue and will be eventually served. Hence, *Theorem 2* is proved.

5. Performance Analysis

5.1. Message Complexity

Best case: The best case will occur when an idle token holder node requests for *CS* and enters *CS*. In this case, no message will be exchanged.

Worst Case: In the worst case, node *i* will contain all other nodes except itself in its request set, therefore, ($n-1$) request messages will be sent. Hence, $n-1$ request messages and one token message (total *n* messages) will be required.

Average Case*:* On an average, the size of the request set varies between 0 to $n-1$.

Average size of request set = $(0+ (n-1)) /2$

Number of messages in average case = $(n-1)/2+1$ (token)= $(n+1)/2$

5.2. Average Waiting Time

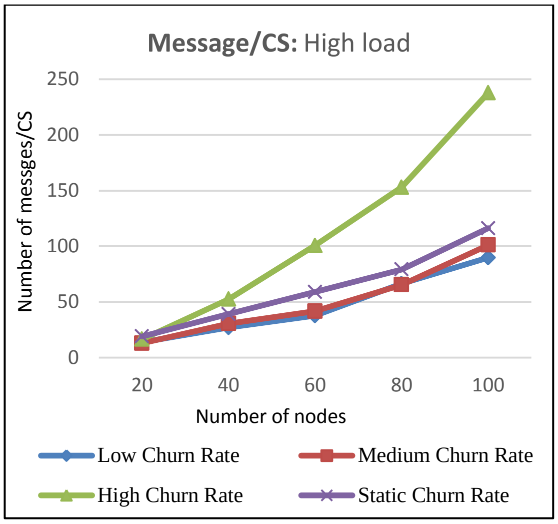
In mutual exclusion, response time is considered under light load conditions. Light load signifies a node holding an idle token. In case, node *i* requests for *CS* to the nodes present in its request set, out of which one can be a token holder. If *T* is the time to transfer the request, another *T* units will be taken to transfer the token to the node *i*. Therefore, *2T* is the average waiting time for best case.

5.3. Synchronization Delay

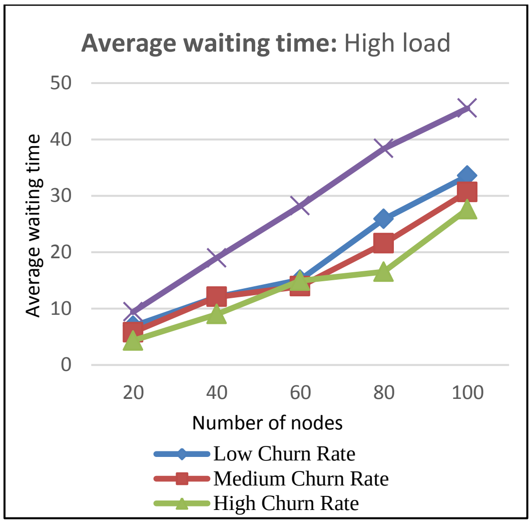
Synchronization delay is considered under heavy load conditions where the token queue is always full of requests. Therefore, when a token holder node comes out of the *CS*, it transfers the token immediately to the node at the front of the token queue (in time *T*) which in turn enters the *CS* immediately. Hence, the heavy load synchronization delay of the algorithm is *T*.

6. Simulation of Algorithm

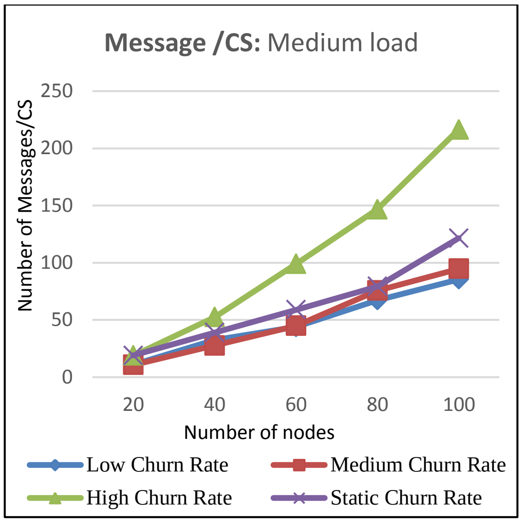
To analyze and study the performance of *DRS* based mutual exclusion algorithm for *MANET* dynamically, *NS2* (discrete event driven network simulator) has been used. The time a process spends in *CS* is assumed to be exponentially distributed with mean value of *500* milliseconds (μ_{cs}). Moreover, the idle time of a node is also considered to be exponentially distributed with mean value (μ_{ncs}). The contention level can be calculated by using the following formula *contention level* = $(\mu_{cs} / (\mu_{cs} + \mu_{ncs}))*100$ [17]. In order to achieve the desired level of contention, the value of μ_{ncs} has been varied. The experiments have been conducted under heavy load (contention level *98%*), medium load (contention level *50%*), and light load (contention level *2%*) conditions. In order to analyze the effect of mobility on the performance of the algorithm, the algorithm has been simulated under light, medium and heavy mobility. The total number of nodes in the ad hoc region is varied in the range of 20 to 100.



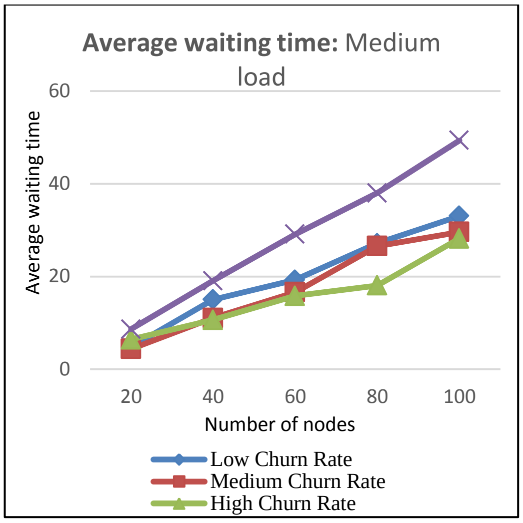
(a)



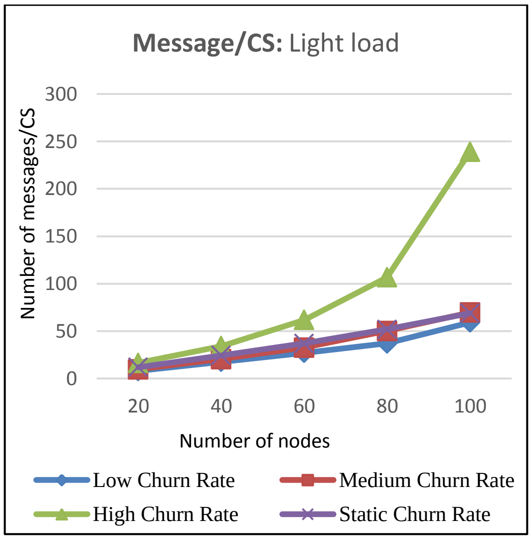
(b)



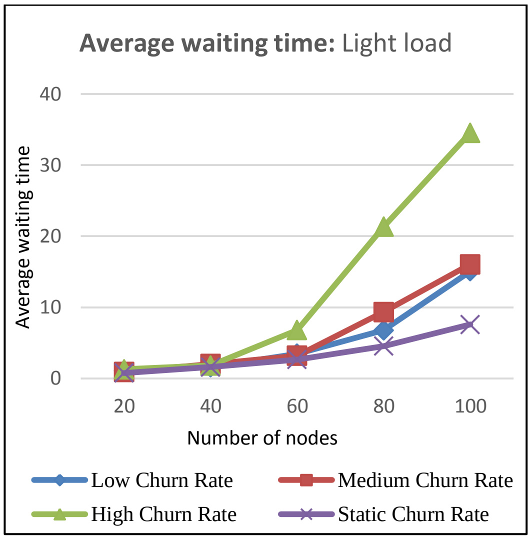
(c)



(d)



(e)



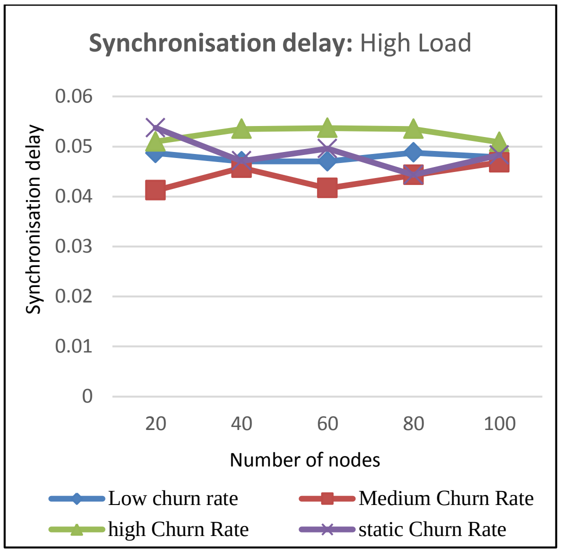
(f)

Fig.2. (a) Messages/CS Vs No. of nodes (High Load) (b) Average waiting time Vs No. of nodes (High Load) (c) Messages/CS Vs No. of nodes (Medium Load) (d) Average waiting time Vs No. of nodes (Medium Load) (e) Messages/CS vs No. of nodes (Light Load) (f) Average waiting time vs No. of nodes (Light Load)

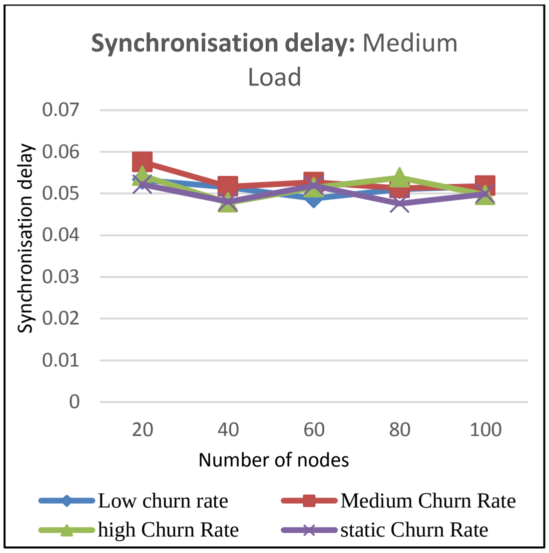
On analyzing the simulation results (fig. 2), it can be said that except in light load, the waiting time of our algorithm for *MANETs* reduces in comparison to static algorithm. Moreover, under the light load condition, the waiting time of mobile *DRS* is slightly higher to the static one under the medium and low churn rate, however, this difference is significant in case of high churn rate. The most striking feature of our algorithm is that under medium load and high load conditions, the algorithm performs significantly better in comparison to static version as far as waiting time is concerned. Moreover, the waiting time reduces as the churn rate is increased. The reason behind this performance benifit is that when a token holder node moves out of neighborhood it broadcasts a message regarding this along with the information about new token holder. Upon receiving this information, a node removes all other nodes from its request sets and add the new token holder in its request set. Now, the probablity that a token holder node wants to move out increases with load and churn rate. Hence, the algorithm performs better in medium/high load and high churn rate conditions as far as waiting time is concerned.

As expected the number of messages per *CS* in mobile *DRS* varies lineraly with the number of nodes in the system. The mobile *DRS* algorithm outperforms the static version algorithm in all the cases except when the churn rate is high (under low load, medium load and heavy laod). The message complexity is reduced because of more updated request sets because of the information broadcasted with *i_am_leaving* message. However, under high churn rate, it is highly probable that a node moves out after sending its request to all nodes in its request set. These messages will be counted whereas the node will not enter *CS*, hence, the message complexity in high churn rate increases.

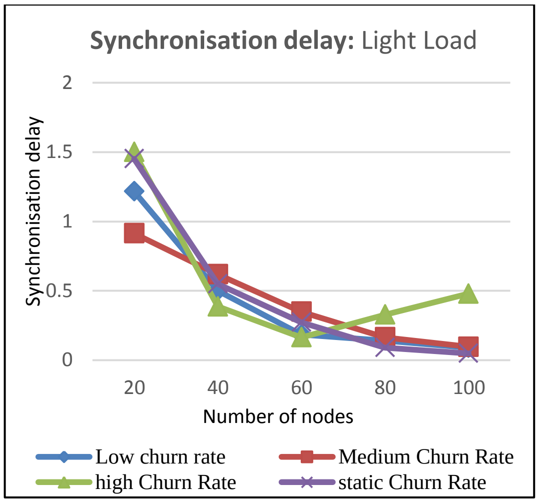
Synchronization Delay. It is customary to analyze synchronization delay under heavy load conditions. In these conditions, the token queue will never be empty. As shown in fig. 3, in light load conditions synchronisation delay varies abruptly as the number of nodes is changed. It is higher when the number of nodes is less in number and reduces as the number of nodes is increased. When we analyse the synchronisation delay in medium and heavy load conditions, it is evident from the graphs that it remains constant in both cases i.e. around *0.05* which we also have taken in case of simulation as the average propagation delay. Hence, it confirms our algorithm according to which the heavy load synchronization delay should be *T*.



(a)



(b)



(c)

Fig.3. (a) Synchronization Delay Vs No. of nodes (High Load) (b) Synchronisation Delay Vs No. of nodes (Medium Load) (c) Synchronization Delay Vs No. of nodes (Light Load)

7. Handling Token Loss

The major challenge in token-based algorithms is to handle the token loss problem. Therefore, here in this section, a fault tolerant version of the exposition has been proposed which is can handle the problem of token loss. Although, it has been assumed that node *i* will not fail and token may loss only in transit. Following features have also been added in the presented algorithm to make it fault tolerant.

1. Token acknowledgment will also be sent by the new token holder node to the previous token holder node i.e. sender.
2. The previous token holder i.e. sender will maintain a copy of the old token till it receives the token acknowledgement.

Following cases are possible when node *i* (sender) will send the token to the node *j* (receiver).

Case (a) Node *j* left the ad hoc region before receiving the token.
In this case, node *j* before leaving the ad hoc region will broadcast the *I_am_leaving* message. When node *i* will receive the *I_am_leaving* message then node *i* (sender) will take the responsibility of token holder node using the copy of stored old token and will hold the idle token (in case, token queue becomes empty after deleting the request of node *j*) or it will transfer token to the next node at the front of token queue and also will wait for the token acknowledgement.

Case (b) In case, node *i* has left before receiving the token acknowledgment from the ad hoc environment.
In case b, node *i* will broadcast *I_am_leaving* message with old token information to all the nodes. Now further two cases are possible in case of *case (b).*

Case (b1) node *j* has received the token and has sent token acknowledgement to the sender node.
In case b1, when node *j* will receive the *I_am_leaving* message of node *i* (sender) then node *j* (receiver) will broadcast a *token alive message* to all the nodes. On receiving the message, all the nodes will delete the stored old token information.

Case (b2) node *j* has left the ad hoc environment.

In case b2, all nodes in the region will receive the broadcasted *I_am_leaving* message of node *j*. As all the nodes have the copy of old token, further two cases are possible in this scenario (1) The token queue is not empty. In this case, the next node at the front of the token queue will declare itself as the token holder. (ii)The token queue is empty. In this case, the lowest id node of the region (decided by the link level protocol of the system) having the copy of old token will become the new token holder. In both the discussed cases, the new token holder broadcasts the new token information message to all nodes. After receiving information about the new token holder, all nodes will delete the old token information.

8. Conclusion and Future Scope

In the present paper, a token-based mutual algorithm using dynamic request set has been proposed. The simulation result shows and validate that the dynamic request set concept can be applied more effectively in ad hoc environment in comparison to static distributed system. The algorithm achieves optimum heavy load synchronization delay of *T* (where *T* is the maximum message propagation delay). Performance analysis of the presented algorithm has also been shown. The solution to token loss problem is also included in the presented exposition. A fully fault tolerant version of the algorithm which is capable of handling message loss and node crash failure is left as a future work.

References

[1] Kshemkalyani AD, Singhal M. Distributed Computing Principles, Algorithms, Systems. Cambridge University Press; 2009.

[2] Dijkstra E W. Hierarchical Ordering of Sequential Processes. Acta Informatica 1971; 115-138.

[3] Chandy KM, Mishra J. The drinking philosopher's problem. ACM transactions on programming languages and systems, Vol. 6, no.4, 1984; 632-646.

[4] Swaroop A. Efficient Group Mutual Exclusion Protocols for Message Passing Distributed Computing System doctoral diss., NIT, Kurukshetra, India; 2009.

[5] Weigang W, Jiannong Ca, Jin Y. A Fault Tolerant mutual exclusion algorithm for mobile ad hoc networks, Pervasive and Mobile Computing, Vol. 4, no. 1, 2008; 139-160.

[6] Suzuki I, Kasmi T. A distributed mutual exclusion algorithm. ACM transactions on computer systems, Vol. 13, no. 4, 1985; 344-349.

[7] Chang YI, Singhal M, Liu MT. A dynamic token-based distributed mutual exclusion algorithm. In the proceedings of the 10th annual international phoenix conference on computers and communications, 1991; 240-24.

[8] Singhal M. A heuristically-aided algorithm for mutual exclusion in distributed systems. IEEE transactions on computers, Vol. 38, no. 5, 1989; 651-662.

[9] Badrinath BR, Acharya A, Imielinski T. Structuring Distributed Algorithms for Mobile Hosts. Proceedings of the 14th International Conference Distributed Computing Systems, 1994; 21–28.

[10] Walter JE, Kini S. Mutual exclusion on Multi–Hop, Mobile Wireless Networks", Technical Report, Texas A&M University, College Station, TX 77843–3112 TR97–014.; 1997.

[11] Walter JE, Welch J, Vaidya NH. A Mutual Exclusion Algorithm for Ad Hoc Mobile Networks, Wireless Networks, Vol. 7, 2001; 585–600.

[12] Baldoni R, Virgillito A, Petrassi R. A Distributed Mutual Exclusion Algorithm for Mobile Ad Hoc Networks. Proceedings of the 7th IEEE International Symposium Computers & Communications, 2002; 539–544.

[13] Kakugawa H and Yamashita M. Self-Stabilizing Local Mutual Exclusion on Networks in which Process Identifiers are not Distinct, in proceeding of 21st symposium on Reliable Distributed Systems (SRDS),

2002; 202-211.

[14] Beauquier J, Datta AK, Gradinariu M, Magniette F. Self-Stabilizing Local Mutual Exclusion and Daemon Refinement, in Proceedings of 13th International symposium on Distributed computing DISC 2000; 223-227.

[15] Attiya H, Kogan A, Welch JL. Efficient and Robust Local Mutual Exclusion in Mobile Ad hoc Networks, IEEE Transactions on Mobile Computing, Vol. 9(3), 2010; 361–375.

[16] Kogan A. Efficient and Robust Local Mutual Exclusion in Mobile Adhoc Networks, Masters Research thesis, Technion- Israel Institute of Technology, Israel; 2008.

[17] Khanna A, Singh AK, Swaroop A.A Leader-Based *k*-Local Mutual Exclusion Algorithm Using Token for MANETs. J. Inf. Sci. Eng. 30(5), 2014; 1303-1319.

[18] Khanna A, Singh AK, Swaroop A. Token Based Group Local Mutual Exclusion Algorithm in MANETs.In: Proceedings of Intelligent Computing, Communication and Devices (ICCD), Advances in Intelligent Systems and Computing (Springer) 309, 2014; 105-113.

[19] Khanna A, Singh AK, Swaroop A. Token Based Group Local Mutual Exclusion Algorithm in MANETs. Recent Development in Wireless Sensor and Ad-hoc Networks: 978-81-322-2128-9; 2014.

[20] Wu AW, Cao J, Raynal M. A Generalized Mutual Exclusion Problem and Its Algorithm. AoxueLuo, Weigang Wu, In: International Conference on Parallel Processing, Vol. 42, 2013; 300-309.

[21] Tamhane SA, Kumar M. A Token Based Distributed Algorithm for Supporting Mutual Exclusion in Opportunistic Networks. Pervasive and Mobile Computing, Vol. 8(5), 2012; 795-809.

[22] Parameswaran M, Hota C. Arbitration-based Reliable Distributed Mutual Exclusion for Mobile Ad-hoc Networks. in Proceedings of Modelling & Optimization in Mobile, Ad Hoc & Wireless Networks, Vol. 11, 2013; 380-387.

[23] Mallikarjuna B, Saritha V, Puspalatha C. Fault Tolerant Resource Management with Mutual Exclusion Algorithm for Mobile Adhoc Networks. International Journal of Scientific and Research Publication, Vol. 2(4), 2012; 1-6.

[24] Karthik R, Gopinath B. Efficient Algorithms for Distributed Mutual Exclusion in Mobile Ad-hoc Network. Indian Journal of Computer Science and Engineering (IJCSE), Vol. 2(6), 2012; 873-884.

[25] Hosseinabadi G, Vaidya NH. Exploiting Opportunistic Overhearing to Improve Performance of Mutual exclusion in Wireless Ad Hoc Networks. in proceeding of Int. conf. on wired/wireless Internet Communications (WWIC), 2012; 162-173.

[26] Itriq M, Dbabat W, Sharieh P. Adaptive Dynamic Resource Synchronization Distributed Mutual Exclusion Algorithm, Journal of Theoretical & Applied Information Technology, Vol. 53(3), 2013; 392-401.

[27] Weigang W, Jiannong C, Yang J. A Fault Tolerant mutual exclusion algorithm for mobile ad-hoc networks. Pervasive and Mobile Computing Vol. 4(1), 2008; 139-160.

[28] Wu W, Zhang J, Luo A, Cao J. Distributed Mutual Exclusion Algorithms for Intersection Traffic Control. IEEE Transactions on Parallel and Distributed System Vol. 26(1), 2015; 65-74.

Authors' Profiles

Ashish Khanna received the Btech degree in CSE from GGSIPU Delhi, India (2004) and M-Tech in IT (2009) from GGSIPU. He is Pursuing his PhD from NIT Kurukshetra. He is presently working as Assistant Professor in CSE Dept. of Maharaja Agrasen Institute of Technology under GGSIPU in Distributed Systems. His research interest includes Distributed Systems, Resource allocation and MANETs.

Awdhesh Kumar Singh received B.E. (CSE) from Gorakhpura (1988). M.E. and Ph.D. (Engg) from Jadavpur University, Kolkata. He is Professor in CSE dept. NIT, Kurukshetra. His research interests are distributed Systems and its variants.

Abhishek Swaroop received the B.Tech in CSE from G.B.P. U, Pantnagar, (1992), M.Tech. in IT from PU Patiala (2004), and Ph.D. from NIT Kurukshetra (2011). He is Professor in CSE Dept., in Galgotias University U.P. His research interests are Distributed Systems and its variants.

How to cite this paper: Ashish Khanna, Awadhesh Kumar Singh, Abhishek Swaroop,"Dynamic Request Set based Mutual Exclusion Algorithm in MANETs", IJWMT, vol.5, no.4, pp.1-14, 2015.DOI: 10.5815/ijwmt.2015.04.01