A DISTRIBUTED MUTUAL EXCLUSION ALGORITHM FOR MOBILE AD HOC NETWORKS

Kayhan Erciyes[1] and Orhan Dagdeviren[2]

[1]Computer Engineering Department, Izmir University, Izmir, Turkey
kayhan.erciyes@izmir.edu.tr
[2]International Computing Institute, Ege University, Izmir, Turkey
orhan.dagdeviren@ege.edu.tr

ABSTRACT

We propose a distributed mutual exclusion algorithm for mobile ad hoc networks. This algorithm requires a ring of cluster coordinators as the underlying topology. The topology is built by first providing clusters of mobile nodes in the first step and then forming a backbone consisting of the cluster heads in a ring as the second step. The modified version of the Ricart-Agrawala Algorithm on top of this topology provides analytically and experimentally an order of decrease in message complexity with respect to the original algorithm. We analyze the algorithm, provide performance results of the implementation, discuss the fault tolerance and the other algorithmic extensions, and show that this architecture can be used for other middleware functions in mobile networks.

KEYWORDS

Mobile Ad hoc Networks, Distributed Mutual Exclusion, Layered Architecture, Clustering, Backbone Formation.

1. INTRODUCTION

Mobile ad hoc networks (MANETs) do not have fixed infrastructure and consist of mobile wireless nodes that have temporary interconnections to communicate over packet radios. The facility that ensures that only one node is in its critical section(CS) at any given time, namely mutual exclusion in a distributed system such as a MANET has received attention from various researchers recently. In general, distributed mutual exclusion algorithms may be classified as permission based or token based. In the permission-based algorithms, a node would enter a CS after receiving permission from all of the nodes in its set for the CS. In token-based algorithms however, the possession of a system-wide unique token would provide the right to enter a CS. Examples of permission-based distributed mutual exclusion algorithms are Lamport's algorithm [1] (3(N-1) messages), Ricart-Agrawala (RA) algorithm (2(N-1) messages) [2] and Maekawa's algorithm [3]. Susuki-Kasami's algorithm [4] (N messages) and Raymond's tree based algorithm [5] (log(N) messages) are examples of token based mutual exclusion algorithms. Safety, liveness and fairness are the main requirements for any mutual exclusion algorithm. Lamport's algorithm and RA algorithm are considered as fair distributed mutual exclusion algorithms in literature. For MANETs, a fault tolerant distributed mutual exclusion algorithm using tokens is discussed in [6] and a k-way mutual exclusion algorithm for ad hoc wireless networks where there may be k processes executing a CS at any time is presented in [7].

In this study, we propose a distributed mutual exclusion algorithm (Mobile_RA) for MANETs based on RA algorithm which requires a backbone topology in the network. The backbone consists of clusters, *coordinators* of which are connected as a ring. In order to realize this topology, a clustering algorithm at the lowest layer provides dynamic clusters of the MANET, using the previously designed Merging Clustering Algorithm (MCA) [8]. The Backbone

Formation Algorithm (BFA) [9] at the second layer provides a virtual ring architecture of the *coordinators* of the clusters formed by MCA.

We briefly review the operation of these two layers and show the design and implementation of the distributed mutual exclusion algorithm which uses the virtual ring structure. Each *coordinator* on the ring performs the required CS entry and exit procedures for the nodes they represent. Using this architecture, we improve and extend the RA algorithm described in [10] to MANETs and show that the algorithm proposed achieves an order of magnitude reduction in the number of messages required to execute a CS at the expense of increased response times and synchronization delays. This significant message decrease will be very convenient in environments that use wireless sensor networks where energy efficiency, therefore message complexity is of paramount importance. The rest of the paper is organized as follows. Section 2 provides the background on clustering algorithms and the backbone formation algorithm. The related work on mutual exclusion in general and in MANETs is described in Section 3. Section 4 describes the proposed algorithm in detail with its proof of correctness and complexity analysis. The implementation results are explained in Section 5, the further extensions are discussed in Section 6, and the conclusions are outlined in Section 7.

2. Background

2.1. Clustering of the MANETs

Clustering is a fundamental approach to manage the MANET services. In clustered networks, nodes are either classified as cluster members, cluster heads or optionally cluster gateways. A cluster member is an ordinary cluster node which sends its request to its cluster head. A cluster head is responsible for managing intra cluster requests and participating in inter cluster operations. The most significant benefit of clustering is maintaining the routing infrastructure for multi hop and multicast communications. Furthermore the network load is distributed more balanced in clustered networks compared to the networks with no infrastructure. Lastly, the clustering method supports a hierarchical management scheme which upper layers can take advantage of it. There are two fundamental graph theoretic approaches for clustering in MANETs. The first approach is the dominating set based clustering in which cluster heads are selected as the elements of the dominating set. Other approach is constructing a spanning tree in which single tree or multiple trees are built for communication.

2.2. Clustering using Dominating Sets

In a graph G, a set $S \subseteq V(G)$ is a dominating set if a vertex is in S or has a neighbor in S where V is the set of vertices. The minimum size of the dominating set in G is called as the domination number. The dominating set S is a connected dominating set if the elements are connected, independent dominating set if the elements are independent, total dominating set if the set has no isolated vertex [11]. The MANET researchers propose different types of dominating sets [12-14] but specifically focus on minimum connected dominating set construction [15-17] which is an NP-hard problem.

2.3. Clustering using Spanning Tree

A tree is a connected acyclic graph. A spanning subgraph $G`$ of G is a subgraph that has same vertex set with G. A spanning tree is a spanning subgraph that has the tree properties. The minimum spanning tree (MST) is a spanning tree with minimum edge weights [11]. The MCA we have designed partitions a MANET into non overlapping balanced clusters and constructs a tree based routing infrastructure. The upper and lower bound parameters are used for balancing clusters. The nodes that have strong communication links with each other are included in same clusters. The main target of the MCA is to decrease the routing cost and to maintain a robust architecture for upper layers. MCA is not a solely routing algorithm, it is a clustering algorithm located on top of any ad hoc routing algorithm. MCA is stable against varying mobility and density conditions shown in [8]. Fig. 1 shows a mobile network of 20 nodes which is partitioned into 4 clusters using MCA. Nodes 14, 18, 17 and 19 are the cluster leaders of the clusters 1, 2, 3 and 4 respectively.

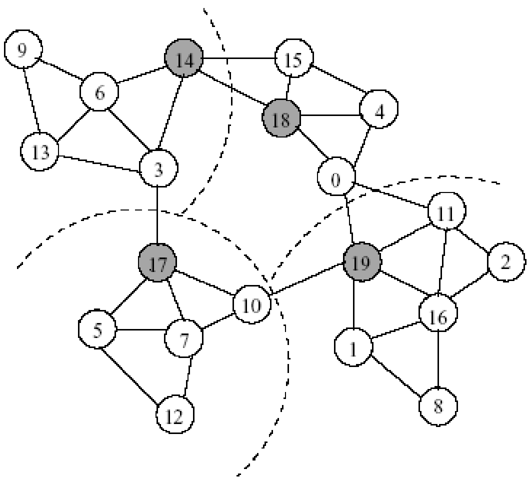

Figure 1. Clustering of the Mobile Network

2.4. Backbone Formation in MANETs

The backbone in wireless ad hoc networks is a path connecting cluster heads that supports a network-wide infrastructure for routing and inter cluster operations. There are energy-efficient [18], multicast oriented [19], tree based [20] and dominating set based [21] backbones which are proposed for MANETs in literature. BFA we have designed [9] constructs a directed ring backbone to give better services for upper layer and to maintain an infrastructure for distributed algorithms running on ring. When BFA is used on top of a balanced clustering algorithm like MCA, a robust infrastructure is constructed for upper layers by supporting load balancing and reducing routing delay. BFA is a semi-distributed algorithm in which cluster heads food their information to the network to maintain a global knowledge of each other. The ordinary cluster members act as router nodes during this operation. The algorithm has two modes of operation; hop-based and position-based backbone formation. The information flooded by cluster heads depends on the mode of operation. In hop-based BFA, number of hops between cluster heads is considered for ring construction. Since cluster heads must share the same information, an agreement must be made between them in highly mobile scenarios. In position-based BFA, each cluster head floods its position to the network. BFA is stable against varying mobility and density conditions shown in [9]. Fig. 2 shows the directed ring architecture formed by the nodes 19, 14, 17 and 18 in a mobile network of 20 nodes which is partitioned into clusters.

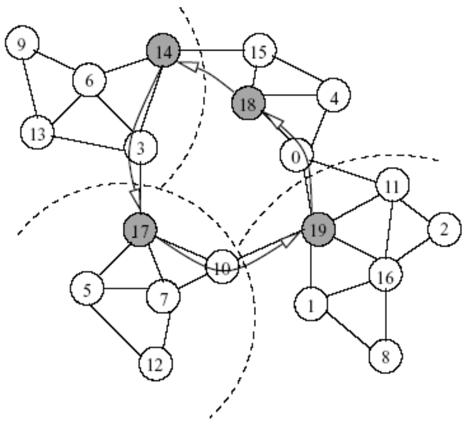

Figure 2. Backbone Formation in Mobile Ad hoc Network

2.5. The Proposed Architecture

The proposed architecture with four layers for distributed mutual exclusion in mobile ad hoc networks is shown in Fig. 3. Distributed applications can be implemented on top of these layers. The lowest layer is the routing layer which can be any ad hoc network routing protocol like AODV [22], DSR [23] or DSDV [24].

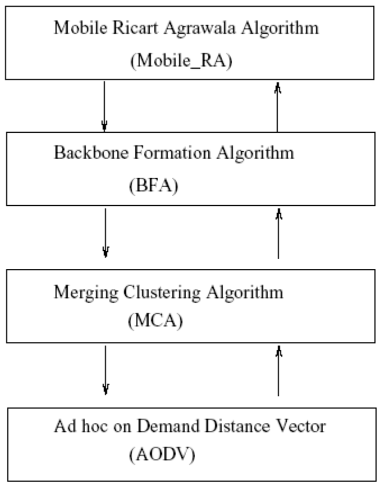
Mobile Ricart Agrawala Algorithm
(Mobile_RA)
Backbone Formation Algorithm
(BFA)
Merging Clustering Algorithm
(MCA)
Ad hoc on Demand Distance Vector
(AODV)

Figure 3. The Proposed Architecture

In this study, we used AODV since it is a widely used routing protocol which also has a stable *ns2* [25] release [26]. The second layer is the clustering layer in which we can use MCA. In the third layer or backbone formation layer, BFA is used that inputs these clustersand builds a directed ring of the clusterheads. BFA can be implemented on top of any clustering algorithm that produces one clusterhead for each cluster. MCA and BFA handle the link failures caused by the mobility of the nodes and also produce stable topologies against varying node densities. Finally the last layer is the Mobile_RA for distributed mutual exclusion. Any distributed algorithm running on top of a ring architecture can be implemented as the last layer.

2.6. Performance Metrics

Performance of a distributed mutual exclusion algorithm depends on whether the system is lightly or heavily loaded. If no other process is in the CS when a process makes a request to enter it, the system is lightly loaded. Otherwise, when there is a high demand for the CS which results in queueing up of the requests, the system is said to be heavily loaded. The important metrics to evaluate the performance of a mutual exclusion algorithm are the number of messages per request, response time and the synchronization delay as described below:

- Number of Messages per Request(M): The total number of messages required to enter CS is an important and useful parameter to determine the required network bandwidth for that particular algorithm. M can be specified for high load or light load in the system as M_{heavy} and M_{light} .

- Response Time(R): The Response Time R is measured as the interval between the request of a node to enter a CS and the time it finishes executing the CS. When the system is lightly loaded, two message transfer times and the execution time of the CS success resulting in $R_{light} = 2T + E$ units. Under heavy load conditions, assuming at least one message is needed to transfer the access right from one node to another, $R_{heavy} = w(T + E)$ where w is the number of waiting requests.

- Synchronization Delay(S): The synchronization delay S is the time required for a node to enter a CS after another node finishes executing it. The minimum value of S is one message transfer time T since one message success to transfer the access rights to another node. The lower bounds for M, R and S are shown in Fig. 4.

M_{light}	M_{heavy}	R_{light}	R_{heavy}	S
3	3	$2T + E$	$w(T + E)$	T

Figure 4. Lower Bounds for Performance Metrics

2.7. Token-Based Algorithm

The general Token Passing(TP) Algorithm for mutual exclusion is characterized by the existence of a single token where the possession of it denotes permission to enter a CS [10]. The token circulation can be performed in a logical ring structure or by broadcasting [4]. In a ring based TP Algorithm, any process that requires its CS will block the token and issue it when it finishes executing. Fairness is ensured in this algorithm as each process waits at most N - 1 entries to enter the CS. There is no starvation since passing is in strict order. The main difficulties with TP Algorithm are as follows. There would be the idle case of no processes entering CS which would incur overhead of constantly passing the token. There could be lost tokens which would require diagnosis and creating a new token by a central node or distributed control is needed and to prevent duplicate tokens, central *coordinator* should ensure generation of only one token. Crashes should also be dealt with as these would require detection of the dead destinations in the form of acknowledgements. One important design issue with TP Algorithm is the determination of the holding time for unneeded token. If this time is too short, there will be high overhead. However, keeping this time too long would result in high CS latency. Performance metrics for TP is given in Fig. 5.

M_{light}	M_{heavy}	R_{light}	R_{heavy}	S
N	N	$2T+E$	$w(T+E)$	T

Figure 5. Performance Metrics of General Token-Based Algorithms

2.8. Ricart Agrawala Algorithm

The RA represents a class of decentralized, permission based mutual exclusion algorithms. In RA Algorithm, when a node wants to enter a CS, it sends a timestamped broadcast *Request* message to all of its peers in that CS request set. When a node receives a *Request* message, it returns a *Reply* message if it is not in the CS or requesting it. If the receiving node is in the CS, it does not reply and queues the request.

However, if the receiver has already made a request, it compares the timestamp of its request with the incoming one and replies the sender if the incoming request has a lower timestamp. Otherwise, it queues the request and enters the CS. When a node leaves its CS, it sends a reply to all the deferred requests on its queue which means the process with the next earliest request will now receive its last reply message and enter the CS. The total number of messages per CS is 2(N - 1) as (N - 1) requests and (N - 1) replies are needed. One of the problems with this algorithm is that if a process crashes, it fails to reply which is interpreted as a denial of permission to enter the CS, so all other processes that want to enter are blocked. Also, the system should provide some method of clock synchronization between processes. The performance metrics for the RA Algorithm are shown in Fig. 6. When a node finishes execution of a CS, one message is adequate for a waiting node to enter, resulting in $S = T$.

M_{light}	M_{heavy}	R_{light}	R_{heavy}	S
$2(N-1)$	$2(N-1)$	$2T+E$	$w(T+E)$	T

Figure 6. Performance Metrics of Ricart Agrawala Algorithm

3. RELATED WORK

Distributed mutual exclusion in mobile networks is a relatively new research area. Singhal et al. [27] proposed a concept of look-ahead technique for distributed mutual exclusion which instead of enforcing mutual exclusion among all the sites of a mobile system, enforces mutual exclusion only among the sites which are concurrently competing for the CS, resulting in less message overhead. Mutual exclusion algorithm involves two issues: First is identifying sites which are concurrently competing for CS, and second enforcing mutual exclusion among these sites. Once a site knows all the sites which are concurrently requesting CS, it can use RA method on those sites to enforce mutual exclusion. Walter et al. [6] proposed a mobility aware token based distributed mutual exclusion algorithm which combines ideas from several papers. The partial reversal technique from [28] used to maintain a destination oriented directed acyclic graph (DAG) in a packet radio network when the destination is static, is used in the algorithm to maintain a token oriented DAG with a dynamic destination. Like the algorithms of [5, 29, 30] each node in the algorithm maintains a request queue containing the identifiers of neighboring nodes from which it has received requests for the token. Like Dhamdhere and Kulkarni's algorithm [30], the algorithm totally orders nodes. The lowest node is always the current token holder, making it a sink toward which all requests are sent. Each node dynamically chooses its lowest neighbor as its preferred link to the token holder. Nodes sense link changes to immediate neighbors and reroute requests based on the status of the previous preferred link to the token holder and the current contents of the local request queue. All requests reaching the token holder are treated symmetrically, so that requests are continually serviced while the DAG is being re-oriented and blocked requests are being rerouted. Baldoni [31] et al. proposed a token based distributed mutual exclusion algorithm suited for mobile ad-hoc networks. The algorithm is based on a dynamic logical ring and combines the two families of token based algorithms (i.e., token asking and circulating token) in order to get an optimal number of messages exchanged per CS access under heavy request load. The algorithm aims at maintaining device power consumption as low as possible by reducing the number of hops traversed per CS execution and by not sending any control messages when no processes request the CS. Mobility is addressed by exploiting the information of the routing table in order to send each message to the closest node in terms of number of hops.

The h-out of-k mutual exclusion problem is also known as the h-out of-k resource allocation problem [32]. It concerns with how to control nodes in a distributed system so that each node can access h resources out of totally k shared resources, $l \leq h \leq k$, with the constraint that no more than k resources can be accessed concurrently. Jiang [33] proposed a prioritized distributed h-out of-k mutual exclusion algorithm for MANETs with real-time or prioritized applications. The proposed algorithm is sensitive to link forming and link breaking and thus is suitable for MANETs. The proposed algorithm claims the highest priority first serve property for real-time applications. For non-real-time applications, one may associate the priority with the number of requested resources to achieve the maximum degree of concurrency. Yang [34] proposed a distributed algorithm to solve the mutual exclusion problem in MANETs. The proposed algorithm improves the CS execution time by allowing at most R tokens to be concurrently dispatched, it employs logical ring construction to adapt the token navigation to the system requirements and it is designed with the consideration of the dynamical link formation characteristics in MANETs and is thus suitable for mobile environments. Wu et al. [35] proposed a permission-based MUTEX algorithm for MANETs. In order to reduce the message cost, the algorithm uses the "look-ahead" technique as in [27], which enforces MUTEX only among the hosts currently competing for the CS. The constraint of FIFO channel is also relaxed. The proposed mechanism handles the "doze" mode and "disconnection" of mobile hosts. Using timeout, a fault tolerance mechanism is introduced to tolerate transient link and host failures.

4. MOBILE_RA ALGORITHM

4.1. General Idea and Description of the Mobile_RA Algorithm

The main idea of the Mobile_RA algorithm is to form *coordinators* as interface of other nodes to the ring [10,36,37]. The relation between the cluster *coordinator* and an ordinary node is similar to a central *coordinator* based mutual exclusion algorithm. The FSM for the ordinary node is given in Fig. 7 where its algorithmic representation is given in Alg. 1. Each node is in *N_IDLE* state initially. When a node wants to enter CS, an internal *Request_CS* event occurs, upon this event the node sends a *Node_Req* to its *coordinator* and makes a state transition to *N_WAITRP* state. After the node receives *Coord_Rep* message from the *coordinator*, it makes state transition to *N_EXECS* state and executes CS. When the node finishes the execution of CS, an internal *Release_CS* event occurs and node sends a *Node_Rel* message to its *coordinator*.

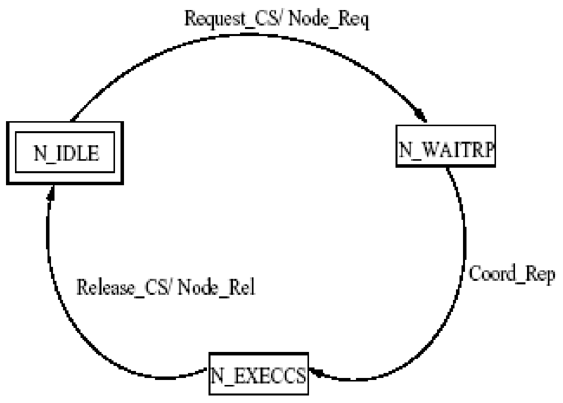

Figure 7. Finite State Machine of the Mobile_RA Node Algorithm

```
1: initially $cur\_state$: the current state of node$_j$
2:      $cur\_state \leftarrow$ N_IDLE
3:      $C$: the coordinator
4:      Legend : □ State $\wedge$ $input\_msg \longrightarrow$ actions
5: loop
6:    □ N_IDLE $\wedge$ $Request\_CS \longrightarrow$ send $Node\_Req$ to $C$
7:      $cur\_state \leftarrow N\_WAITRP$
8:    □ N_WAITRP $\wedge$ $Coord\_Rep \longrightarrow cur\_state \leftarrow N\_EXECCS$
9:    □ N_EXECCS $\wedge$ $Release\_CS \longrightarrow$ send $Node\_Rel$ to $C$
10:     $cur\_state \leftarrow N\_IDLE$
11: end loop
```

Algorithm 1. Mobile_RA's Ordinary Node Pseudocode for Node$_j$ Receiving Message from Node$_i$

The *coordinator* side of the distributed algorithm is more complicated than the ordinary node side. The types of messages exchanged in the *coordinator* side of the algorithm are *Node_Req*, *Coord_Req*, *Coord_Rep* and *Node_Rel* which are described below:

- *Node_Req*: Any node which wants to execute CS, sends a *Node_Req* message to its *coordinator*. After sending this message, node waits for *Coord_Rep* message from its *coordinator* to execute CS.

- *Coord_Req*: When a *coordinator* receives a *Node_Req* message from an ordinary node, it sends *Coord_Req* message to the next *coordinator* on the ring if all pending requests in its cluster have greater timestamps than the timestamp of this *Node_Req* message. Otherwise, it enqueues this message this message to the *wait_queue*. If a *coordinator* receives a *Coord_Req* message, it forwards this message according to the timestamp of this message. If the *coordinator* which is the originator of this message receives its own message, it sends a *Coord_Rep* message if there no other waiting requests with timestamp lower than itself or a CS executing node at the same time.

- *Coord_Rep*: A node which wants to execute CS, will execute CS after receiving this message from its *coordinator*.

- *Node_Rel* : After executing CS, the node sends this message to its *coordinator* indicating that their execution is completed.

The *coordinators* can be either in *IDLE*, *WAITRP* or *WAITND* state as described below:

- *IDLE*: *Coordinators* in *IDLE* state only forwards the *Coord_Req* messages to the next *coordinator* on the ring. If a *Node_Req* message is received, *coordinator* sends a *Coord_Req* message to the next *coordinator* and makes a transition to *WAITRP* state.

- *WAITRP*: *Coordinator* in *WAITRP* state waits for its original *Coord_Req* message. *Coord_Req* messages of other *coordinators* are enqueued if the timestamp is greater. After receiving its original *Coord_Req* message, *coordinator* makes a transition to *WAITND* state. Received *Node_Req* messages are forwarded as *Coord_Req* messages to next *coordinators*.

- *WAITND*: *Coordinator* in *WAITND* state waits for *Node_Rel* message from its cluster member which executes CS. After receiving *Node_Rel*, it makes a transition to *IDLE* state, if there is no pending request from its clustermembers in its *wait_queue*. Otherwise it makes a transition to *WAITRP* state. Received *Node_Req* messages are forwarded as *Coord_Req* messages to the next *coordinators*.

The finite state machine of the Mobile_RA *coordinator* is shown in Fig. 8 [10,36] and its algorithmic representation is given in Alg. 2.

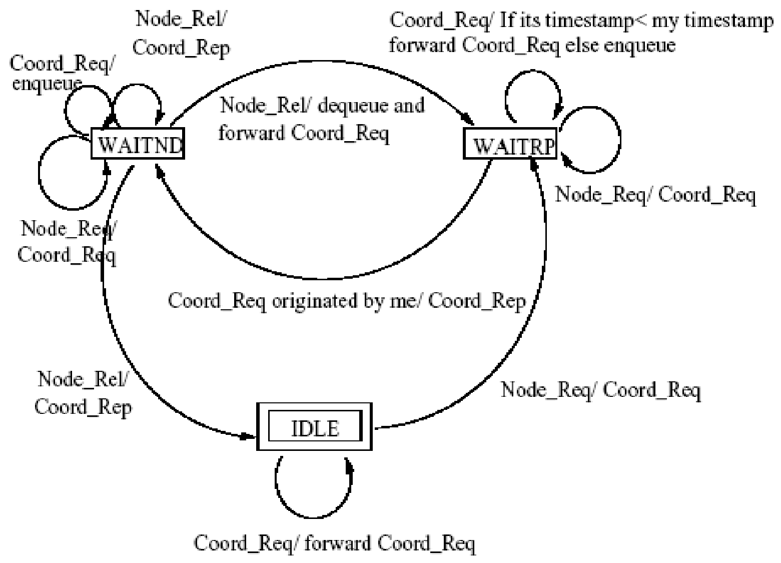

Figure 8. Finite State Machine of the Mobile_RA Coordinator Algorithm

```
initially $cur\_state$: the current state of node$_j$
    $cur\_state \leftarrow$ IDLE
    $NC$: the next coordinator in the ring
    $queue$: the waiting queue, $t$ is the timestamp, $m$ is the type, $s$ is the source, $o$ is the originator of
the request at the top of the queue.
    $msg$: the received message, includes: source ($s$), originator ($o$), timestamp ($t$)
    Legend : □ State $\wedge$ $input\_msg \longrightarrow$ actions
loop
  □ IDLE $\wedge$ $Node\_Req \longrightarrow$ send $Coord\_Req$ to $NC$
    $cur\_state \leftarrow WAITRP$
    enqueue $msg$
  □ IDLE $\wedge$ $Coord\_Req \longrightarrow$ forward $Coord\_Req$ to $NC$
  □ WAITRP $\wedge$ $Node\_Req \longrightarrow$ send $Coord\_Req$ to $NC$
    enqueue $msg$
  □ WAITRP $\wedge$ $Coord\_Req \longrightarrow$ if $msg_o = j$ then
      send $Coord\_Rep$ to $msg_s$
      dequeue
      $cur\_state \leftarrow WAITND$
    else
      if ($msg_t < queue_t$) or ($msg_t = queue_t$ and $j > i$) then
        forward $Coord\_Req$ to $NC$
      else
        enqueue $msg$
      end if
    end if
  □ WAITND $\wedge$ $Node\_Req \longrightarrow$ enqueue $msg$
  □ WAITND $\wedge$ $Coord\_Req \longrightarrow$ enqueue $msg$
  □ WAITND $\wedge$ $Node\_Rel \longrightarrow$ if ($queue_m = Coord\_Req$) and ($queue_o = j$) then
      send $Coord\_Rep$ to $queue_s$
      dequeue
    else
      if $queue$ includes $Node\_Req$ then
        $cur\_state \leftarrow WAITRP$
        while ($queue_m = Coord\_Req$) and ($queue_o \neq j$)
          $mes \leftarrow$ dequeue
          send $mes$ to $NC$
        end while
      else
        $cur\_state \leftarrow IDLE$
      end if
    end if
end loop
```

Algorithm 2. Mobile_RA's Coordinator Pseudocode for Node$_j$ Receiving Message from Node$_i$

4.2. An Example Operation

Fig. 9 shows an example scenario for the Mobile_RA Algorithm. There are 30 nodes in this scenario, and the network is partitioned into 4 clusters. Node 17, node 28, node 29 and node 24 are the clusterheads. The name of a cluster is the id of its clusterhead. Node 15, node 13, node 23 makes request for CS respectively at 4.10s, 4.20s, 4.25s. Execution time of CS is taken as 350ms. The following describes the events that occur:

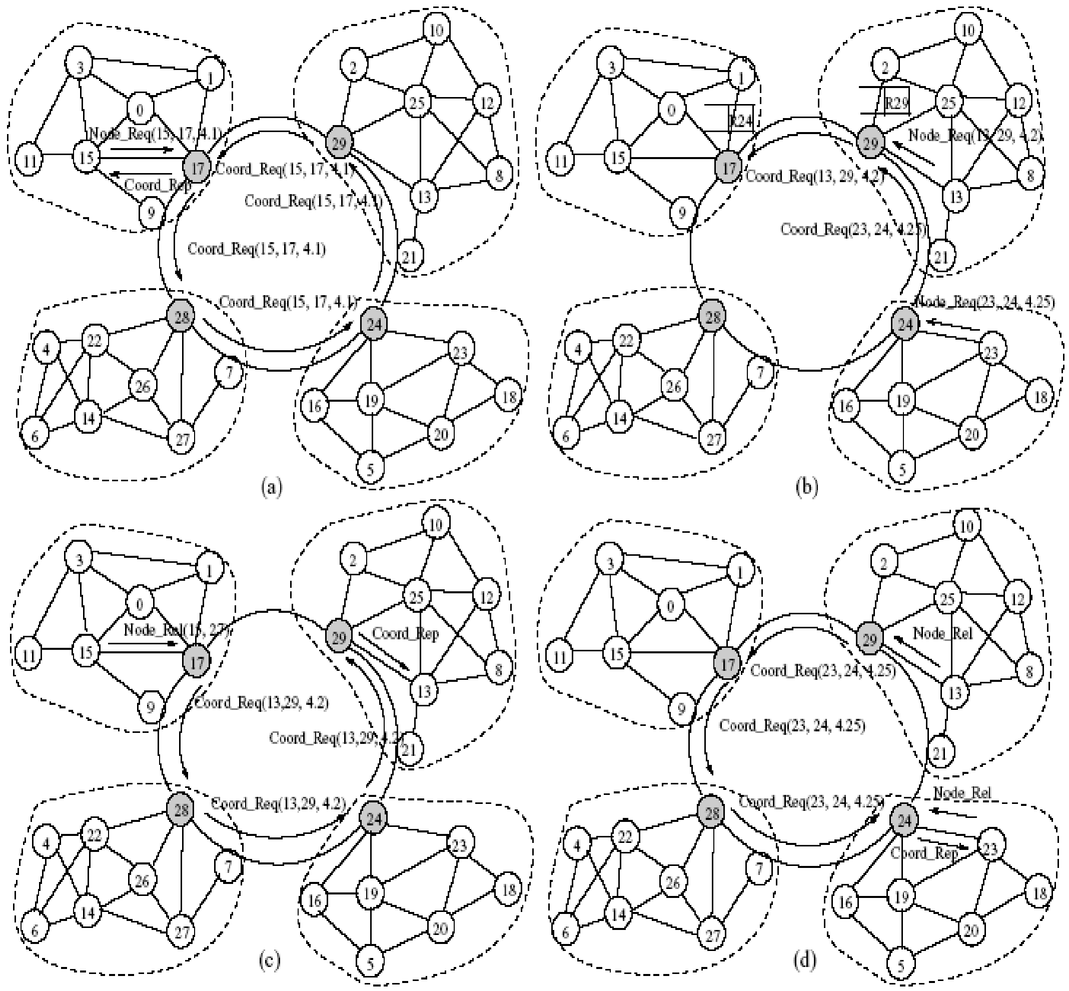

Figure 9. Operation of the Mobile_RA Algorithm

1. Node 15, in cluster 17 makes a CS request at 4.10s by sending a *Node_Req (15, 17, 4.10)* message to node 17 which is the cluster *coordinator*. Node 17 receives the message at 4.11s and changes its state to *WAITRP*. Node 17 sends a *Coord_Req (15, 17, 4.10)* message to the next *coordinator* (node 28) on the ring. Node 28, which is in *IDLE* state and has no pending requests in its cluster, receives the *Coord_Req (15, 17, 4.10)* message at 4.13s and forwards the message to the next *coordinator* (node 24) on the ring. The message traverses the ring and is received by node 17 which is in *WAITRP* state at 4.21s meaning all of the *coordinators* have confirmed that either they have no pending requests or their pending requests all have higher timestamps. Node 17 sends a *Coord_Rep* message to node 15 and changes its state to *WAITND*. Node 15 receives the *Coord_Rep* message at 4.18s and enters the CS. Step 1 is depicted in Fig. 9.(a).

2. Node 13, in cluster 29 makes a CS request by sending a *Node_Req (13, 29, 4.20)* at 4.20s. Node 29 receives the *Node_Req (13, 29, 4.20)* message at 4.21s and sends a *Coord_Req (13, 29, 4.20)* message to its next *coordinator* (node 17) on the ring. Node 17, which is in *WAITND* state, receives the message and enqueues the *Coord_Req (13, 29, 4.2)* at 4.22s. Node 23 makes a CS request at 4.25s. Node 29 which is in *WAITRP* state receives the *Coord_Req (23, 24, 4.25)* message and enqueues the message at 4.28s. Step 2 is depicted in Fig. 9.(b).

3. Node 15 exits from CS at 4.53s and sends a *Node_Rel* message to node 17. Node 17 which is in *WAITND* state receives the message at 4.54s and makes a transition to *IDLE* state. Node 17 dequeues and forwards *Coord_Req (13, 29, 4.20)* message to the next *coordinator* (node 28). The *Coord_Req (13, 29, 4.20)* message is forwarded by node 24 since its request has higher timestamp. Node 29 receives its original request at 4.60s and sends a *Coord_Rep* message to node 13. Node 13 enters the CS at 4.61s. Step 3 is depicted in Fig. 9.(c).

4. Node 13 finishes to execute CS at 4.96s. Node 29 receives the *Node_Rel* message at 4.97s. Node 29 dequeues and forwards the *Coord_Req (23, 24, 4.25)* message to its next *coordinator* (node 17) on the ring. Operation is continued as explained before. Node 24 receives *Node_Rel* message from node 23 at 5.38s. The Step 4 is depicted in Fig. 9.(d). If there are multiple requests within the same cluster, timestamps are checked similarly for local request. The order of execution in this example is nodes 15 $\rightarrow$ 13 $\rightarrow$ 23 in the order of the timestamps of the requests.

4.3. Analysis

4.3.1. Proof of Correctness

In this section we prove the correctness of Mobile_RA Algorithm by showing the safety, liveness and fairness attributes. We define the terms below:

- N_i : The node whose id equals to i.
- RT_i : The request of N_i where its timestamp is T_i .
- C_i : N_i 's *coordinator* node.

Theorem 1. *In Mobile_RA Algorithm, at most one host can be in the CS at any time (safety attribute).*

Proof. We prove the theorem by contradiction. Assume that two nodes N_i and N_j are executing the CS concurrently. In this situation, *Coord_Req* messages of C_i and C_j must be circulated through the ring and they must be received by their originators concurrently. If $T_i > T_j$ than C_j enqueues *Coord_Req* message of N_i . When C_j receives its own *Coord_Req* message, it sends a *Coord_Rep* message to N_j . After N_j finishes the execution of the CS, it sends a *Node_Rel* message to C_j . When C_j receives this message, it dequeues the *Coord_Req* message of N_i and forwards to C_i where C_i waits for the *Coord_Req* message of N_i . Therefore N_i and N_j can not execute CS concurrently. If $T_i = T_j$ than node ids are used to break symmetry as shown in the algorithmic representation in Alg. 2. In all cases only one node executes CS, the other nodes wait for it.

Theorem 2. *The Mobile_RA Algorithm is deadlock and starvation free (liveness attribute).*

Proof. The proof is by contradiction. Assume that a node (N_i) waits endlessly for *Coord_Rep* message which will never arrive. In this situation, *Coord_Req* message of N_i must be enqueued by some other *coordinator* (C_j) and that message will not be dequeued. If C_j has other requests such that $RT_j < RT_i$, than C_j enqueues the *Coord_Req* message of N_i , but later it dequeues and forwards this *Coord_Req* message when C_j receives *Node_Rel* from N_j . If C_j has no request than it forwards the *Coord_Req* message of N_i . We contradict with our assumption.

Theorem 3. *Each node gets a fair chance to execute CS in Mobile_RA Algorithm where requests are executed in order. (fairness attribute).*

Proof. We prove the theorem by contradiction. Assume that N_j executes the CS earlier than N_i although $T_i < T_j$. In this situation, each *coordinator* should have a empty queue or RT_j should be at the top of the queue. But since C_i enqueued RT_i in its queue and the requests in queue are ordered according to their timestamps from smallest to greatest, we contradict with our assumption.

4.3.2. Performance Metrics

The upper and lower bounds for total message number, response time and synchronization delay are analyzed below [10, 36].

Theorem 4. *The total number of messages per CS using the Mobile_RA Algorithm is k+3d where k is an upper bound on the number of neighbor nodes in the ring including the cluster coordinators and d is an upper bound on the diameter of a cluster.*

Proof. An ordinary node in a cluster requires three messages (*Node_Req*, *Coord_Rep* and *Node_Rel)* per CS to communicate with the *coordinator*. Each of these messages would require maximum d transfers between a node and the *coordinator*. The full circulation of the *coordinator* request (*Coord_Req*) requires k messages resulting in k+3d messages in total.

Corollary 1. *The Synchronization Delay (S) in the Mobile_RA Algorithm varies from 2dT to (k + 2d - 1)T.*

Proof. When the waiting and the executing nodes are in the same cluster, the required messages between the node leaving its CS and the node entering are the *Node_Rel* from the leaving node and *Coord_Rep* from the *coordinator* resulting in $2dT$ message times for S_{min}. However, if the nodes are in different clusters, the *Node_Rel* message has to reach the local *coordinator* in d steps, circulate the ring through k-1 node to reach the originating cluster *coordinator* in the worst case and a *Coord_Rep* message from the *coordinator* is sent to the waiting nodes in d steps resulting in $S_{max}=(k-1)T+2dt=(k+2d-1)T$.

Corollary 2. *In the Mobile_RA Algorithm, the response times are $R_{light}=(k + 3d)T + E$ and R_{heavy} varies from w(2dT +E) to w((k +2d-1)T + E) where k is the number of clusters and w is the number of pending requests.*

Proof. According to Theorem 1, the total number of messages required to enter a CS is k+3d. If there are no other requests, the response time for a node will be $R_{light}=(k + 3d)T + E$ including the execution time(E) of the CS. If there are w pending requests at the time of the request, the minimum value $R_{heavy\ min}$ is $w(2dT + E)$. In the case of S_{max} described in Corollary 1, $R_{heavy\ max}$ becomes $w((k + 2d-1)T + E)$ since in general $R_{heavy}=w(S + E)$.

The comparison of Mobile_RA and RA is shown in Fig. 10. If m is an upperbound on the number of nodes in a cluster and if we further assume k=m, the message complexity of Mobile_RA becomes $\sqrt{N}$ +3d.

	$Mobile\_RA$	RA
Total Message Count	k+3d ($\sqrt{N}$+3d when k=m)	2(N-1)
Response time$_{light}$	(k+3d)T+E	2T+E
Response time$_{heavy}$	w((k-1+2d)T+E)	w(T+E)
Synchronization delay	(k+2d-1)T	NT

Figure 10. Comparison of Mobile_RA and RA

5. Performance Evaluation

We implemented the distributed mutual exclusion algorithm with the *ns*2 simulator. IEEE 802.11 standards are used for medium access control and physical layer where the transmission range of a mobile node is 250m. The clustering algorithm, MCA [8], is modified, tuned and the results are re-measured that means the results in this section are different from the results in [37]. A random load generator is implemented to generate high, medium and low loads for different number of nodes. Total number of nodes are selected from 10 to 50 nodes. We limit the upper bound of the network size to 50 since MANET simulations greater than this size is not very applicable [7, 26, 38-42]. Different sizes of flat surfaces are chosen for each simulation to create very small, small and medium distances between nodes, as well as, high dense, dense and medium connected topologies. Surface areas vary from 120m *x* 600m to 600m *x* 600m, 130m *x* 650m to 650m *x* 650m, 140m *x* 700m to 700m *x* 700m respectively. Average degree of the network is approximately *N*/4 for the medium connected, *N*/3.5 for the dense connected and *N*/3 for the highly dense connected networks where *N* denotes the total number of nodes in the network. Random movements are generated for each simulation and random waypoint model is chosen as the mobility pattern. Low, medium and high mobility scenarios are generated and respective node speeds are limited from 1.0m/s to 5.0m/s, 5.0m/s to 10.0m/s, 10.0m/s to 20.0m/s. Upper bound and lower parameters are changed to obtain different size of clusters.

Response times and synchronization delays as measured with respect to load, mobility, density and number of clusters are recorded. Execution of CS is selected as 100ms. Response time increases linearly with the load as seen in Fig. 11. In low loaded networks, average response times vary from 4,93s to 11,49s whereas in high loaded networks average response times vary from 8,15s to 20,75s as seen in Fig. 11. The synchronization delay is 0 in low load scenarios since there will be no waiting requests in the queues. The synchronization delay values vary from 500ms to 2200ms in medium loaded scenarios as shown in Fig. 12. When the load is increased synchronization delay linearly increases due to waiting time of requests in the queue as seen in Fig. 12. Also response time and synchronization delay increases due to collisions and routing delays caused by high network traffic as shown in Fig. 11 and Fig. 12.

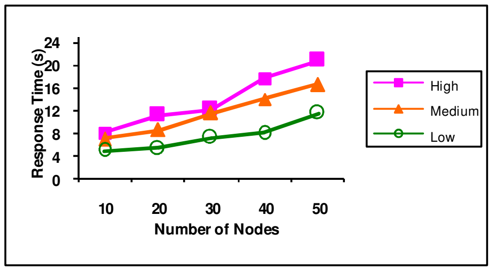

Figure 11. Average Response Time against Load

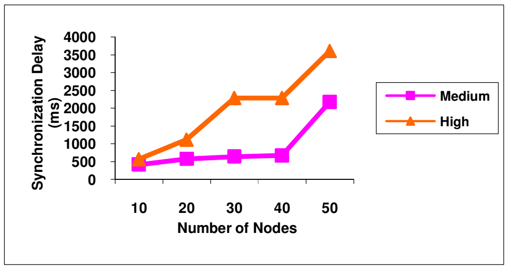

Figure 12. Average Synchronization Delay against Load

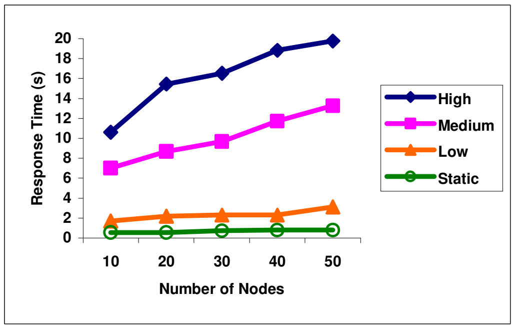

Figure 13. Average Response Time against Mobility

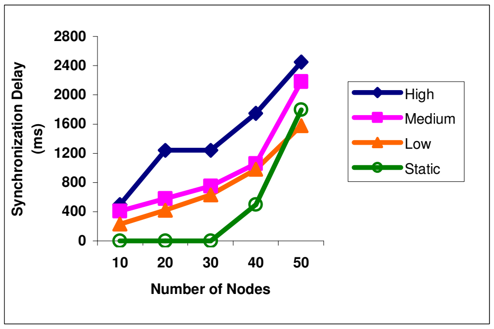

Figure 14. Average Synchronization Delay against Mobility

Response time and synchronization delay increase by the mobility parameter due to the rapid change of network topology as shown in Fig. 13 and Fig. 14. In static networks, the average response time increases from 500ms to 750ms. In high mobile scenarios, due to frequent link changes, the response time values vary from 10s to 20s.

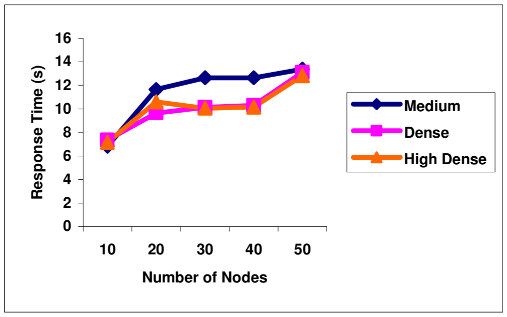

Figure 15. Average Response Time against Density

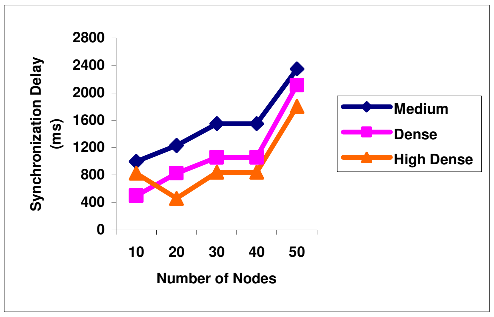

Figure 16. Average Synchronization Delay against Density

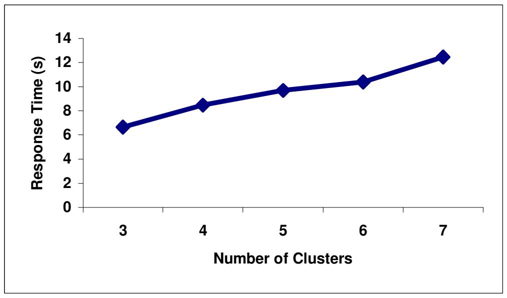

Figure 17. Average Response Time against the Number of Clusters

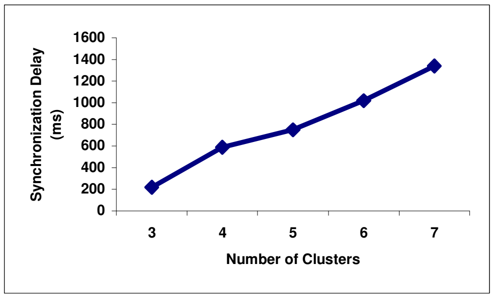

Figure 18. Average Synchronization Delay against the Number of Clusters

Fig. 15 and Fig. 16 show the effect of distance between nodes to response time and synchronization delay. Generally, for the networks with higher connectivity, the average response time and synchronization delay values are 0.1-1s lower. To measure the effect of clustering, cluster upper and lower bound parameters of MCA are adjusted to create 3 to 7 clusters. In fixed number of nodes, as the cluster size increases, total number of clusters in the network decreases. This also reduces the number of cluster leaders forming the ring and routing delay which causes a decrease in the response time and synchronization delay as shown in Fig. 17 and Fig. 18.

Fig. 19 and Fig. 20 show the impact of clustering nodes as opposed to leaving them without a hierarchy. RA algorithm is implemented in MANETs and compared with Mobile_RA. The average response time and synchronization delay values can be seen in Fig. 19 and Fig. 20. The Mobile_RA is far more scalable than RA for MANETs as seen in Fig. 19 and Fig. 20.

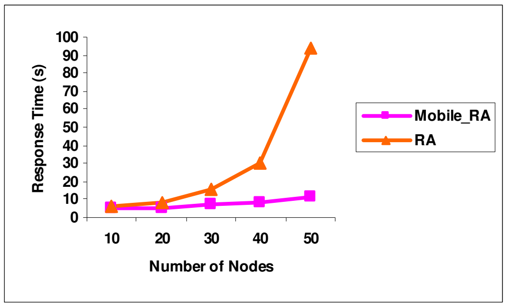

Figure 19. Comparison of Response Times

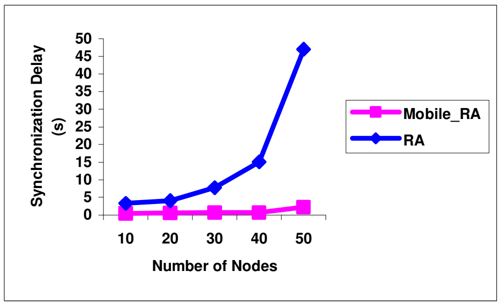

Figure 20. Comparison of Synchronization Delay

Consequently, our results conform to the analysis that response time values against low and medium loads increase linearly with a small gradient. Synchronization delay values against medium and high load also increase linearly. Response time against high load makes a sharp increase due to high network traffic. Synchronization delay values are stable under different mobility and density. Also, response time and synchronization delay values decrease linearly against the number of clusters in MANET. Clustering the network greatly reduces the response time and the synchronization delay values in the flat mobile ad hoc networks.

6. DISCUSSIONS

In the proposed architecture shown in Fig. 3, the link failures caused by the mobility of the nodes are handled by the periodic execution of MCA and BFA. Therefore Mobile_RA does not need to deal with the topological changes. Since MCA and BFA isolates the upper layers from the link failures, other protocols such that Chang and Robert's leader election [43] can be used to select a super leader to provide a central authority [44]. This infrastructure is robust in the case of updating the failed links but failures of the nodes are not covered. For example a *coordinator* may fail during executing Mobile_RA which results to stop the execution of the distributed algorithm. By the addition of an fault tolerance module orthogonal to these architecture, the node failures can be recovered as shown in Fig. 21. This module can take the cluster information from MCA and can choose the backup cluster heads. The other nodes belonging to the same cluster can be informed about the backup cluster head. Each backup polls its cluster head periodically and receives heart beat signals. If a cluster head does not send a heart beat signal for a predefined number of times, than the backup becomes the new cluster head. If a backup does not poll a cluster head for a predefined number of times than the cluster head chooses a new backup. By this way, a recovery mechanism for the crash of a clusterhead and its backup is maintained but if both of them fails at the same time, additional procedures are needed. Fault tolerance module can extract the ring information from BFA where each cluster head may broadcast its backup id to the ring. If a cluster head in the ring crashes, the backup will take the place of the crashed cluster head in the ring.

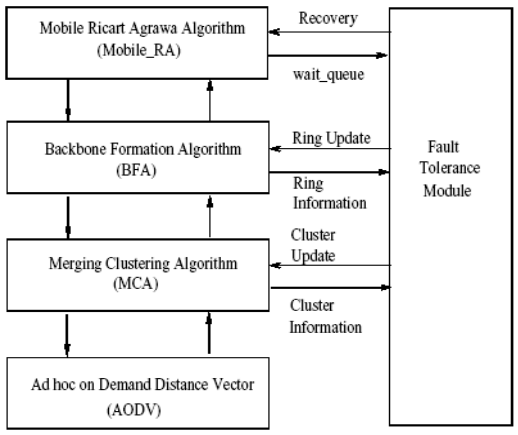
Mobile Ricart Agrawa Algorithm (Mobile_RA)
Recovery
wait_queue
Backbone Formation Algorithm (BFA)
Ring Update
Ring Information
Merging Clustering Algorithm (MCA)
Cluster Update
Cluster Information
Ad hoc on Demand Distance Vector (AODV)
Fault Tolerance Module

Figure 21. Fault Tolerance Module

To provide fault tolerance of the distributed mutual exclusion, the *coordinators* should share its *wait_queue* with their backup nodes. If a *coordinator* is crashed, the backup should have all information of the *coordinator*. The *coordinators* executing Mobile_RA Algorithm sends their *wait_queue* to their fault tolerant module as shown in Fig. 21 when an update in the *wait_queue* occurs. The *wait_queue* is appended to the heartbeat message which is sent periodically to the backup cluster head upon an update. By this way, if a crash occurs in a cluster head, its backup will be the new cluster head, the new cluster head will choose a new backup, the ring will be repaired and since backup is the replica of the cluster head, the distributed mutual exclusion operations may continue to progress. Also the new cluster head will advertise itself to the cluster members. The crashes in the ordinary nodes are easy to handle. If a node crashes during CS execution, the *coordinator* will make a state transition form *WAITND* state to *IDLE* state and continue to process new requests.

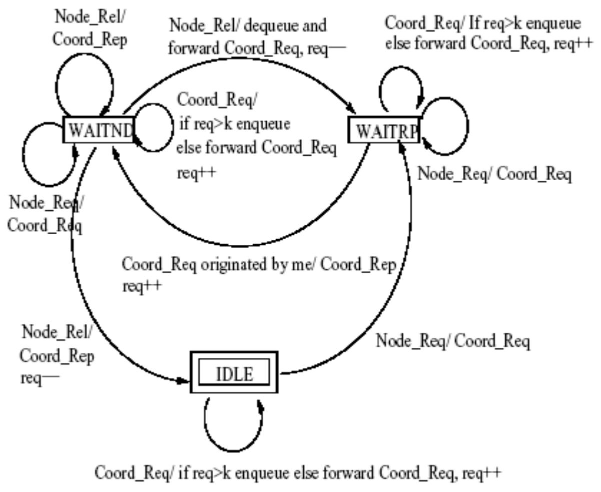
Node_Rel/ Coord_Rep
Node_Rel/ dequeue and forward Coord_Req, req—
Coord_Req/ If req>k enqueue else forward Coord_Req, req++
Coord_Req/ if req>k enqueue else forward Coord_Req req++
WAITND
WAITRP
Node_Req/ Coord_Req
Node_Req/ Coord_Req
Coord_Req originated by me/ Coord_Rep req++
Node_Rel/ Coord_Rep req—
Node_Req/ Coord_Req
IDLE
Coord_Req/ if req>k enqueue else forward Coord_Req, req++

Figure 21. Finite State Machine of the Extended Mobile_RA Algorithm

Mobile_RA can be extended to solve *h*-out of-*k* problem as shown in Fig. 22. A new variable called total number of requests (*req*) is added. When a *coordinator* receives a *Coord_Req* message, if *req* does not exceed *k*, *Coord_Req* message is forwarded else it is enqueued. When a *Node_Rel* message is received by the *coordinator*, *req* is decremented.

7. CONCLUSIONS

We proposed a hierarchical distributed mutual exclusion algorithm for mobile ad hoc networks. The communication infrastructure to run the algorithm consists of a number of clusters of mobile nodes where each cluster is represented by a *coordinator* and the *coordinators* are connected to form a ring. The cluster formation and ring formation can be handled efficiently by the Merging Clustering Algorithm and the Backbone Formation Algorithm. Due to this hierarchical structure, significant gains in total message complexities are obtained with respect to the original Ricart-Agrawala Algorithm as shown in Fig. 6. Equating the specified parameters in fact gives an order of improvement over the classical Ricart Agrawala Algorithm. We also showed experimentally that the response times and synchronization delays conform to theoretical analysis. We discussed the fault tolerance and algorithmic extensions for the proposed architecture. Our work is ongoing and we are planning to implement other distributed system middleware functions such as load balancing [45-47] on top of this layered architecture.

REFERENCES

[1] L. Lamport, (1978) "Time, Clocks and the Ordering of Events in a Distributed System", *Communications of the ACM*, Vol. 2, pp. 558-565.

[2] G. Ricart & A. Agrawala, (1981) "An Optimal Algorithm for Mutual Exclusion in Computer Networks", *Communications of the ACM*, Vol. 24 (1), pp. 9-17.

[3] M. Maekawa, (1985) "A sqrt(n) Algorithm for Mutual exclusion in Decentralized Systems". *ACM Transactions on Computer Systems*, Vol. 3 (2), pp. 145-159.

[4] I. Susuki & T. Kasami, (1985) "A Distributed Mutual Exclusion Algorithm. *ACM Transactions on Computer Systems*, Vol. 3(4), pp. 344-349.

[5] K. Raymond, (1989) "A Tree-based Algorithm for Distributed Mutual Exclusion", *ACM Transactions On Computer Systems*, Vol. 7 (1), pp. 61-77.

[6] J. E. Walter, J. L. Welch & N. H. Vaidya, (2001) "A Mutual Exclusion Algorithm for Ad Hoc Mobile Networks", *Wireless Networks*, 2001, 7 (6), pp. 585-600.

[7] J. E. Walter, G. Cao & M. Mohanty (2001), "A K-way Mutual Exclusion Algorithm for Ad Hoc Wireless Networks", *In Proc. of the First Annual workshop on Principles of Mobile Computing*.

[8] O. Dagdeviren, K. Erciyes & D. Cokuslu D. (2005) "Merging Clustering Algorithms in Mobile Ad Hoc Networks". *In Lecture Notes in Computer Science 3816*, Springer-Verlag, pp. 56-61.

[9] O. Dagdeviren & K. Erciyes, (2006) "A Distributed Backbone Formation Algorithm for Mobile Ad Hoc Networks". *In Lecture Notes in Computer Science,* Vol. 4330, Springer-Verlag, pp. 219-230.

[10]K. Erciyes, (2004) "Cluster-based Distributed Mutual Exclusion Algorithms for Mobile Networks". *In Lecture Notes in Computer Science,* Vol. 3149, Springer-Verlag, pp. 933-940.

[11] D. West, (2001) Introduction to Graph Theory. Second edition, Prentice Hall, Upper Saddle River, N.J.

[12] Y. P. Chen & A. L. Liestman, (2002) "Approximating Minimum Size Weakly-Connected Dominating Sets for Clustering Mobile Ad Hoc Networks". *In the Proc. of 3rd ACM Int. Symp. Mobile Ad Hoc Net. and Comp.*

[13] Y. P. Chen & A. L. Liestman, (2003) "A Zonal Algorithm for Clustering Ad Hoc Networks". *International Journal of Foundations of Computer Science*, Vol. 14(2), pp. 305-322.

[14] I. Stojmenovic, M. Seddigh & J. Zunic, (2002) "Dominating Sets and Neighbor Elimination-Based Broadcasting Algorithms in Wireless Networks". *IEEE Trans. on Parallel and Distributed Systems*, Vol. 13, pp. 14-25.

[15] S. Guha & S. Khuller, (1996) "Approximation Algorithms for Connected Dominating Sets", *Algorithmica*, 1996, Vol. 20, pp. 374-387.

[16] J. Wu & H. Li, (2001) "A Dominating-Set-Based Routing Scheme in Ad Hoc Wireless Networks". *Springer Science+Business Media B. V.*, Formerly Flower Academic Publishers.

[17] D. Cokuslu & K. Erciyes, O. Dagdeviren, (2006) "A Dominating Set Based Clustering Algorithm for Mobile Ad hoc Networks". *In Lecture Notes in Computer Science 3991*, Springer-Verlag, pp. 571-578.

[18] L. Haitao & R. Gupta, (2004) "Selective Backbone Construction for Topology Control in Ad Hoc Networks". *In Proc. of the Intl. Conf. on Mobile Ad-hoc and Sensor Systems*, pp. 41-50.

[19] W. Ya-feng, X. Yin-long, C. Guo-liang, & W. Kun, (2004) "On the Construction of Virtual Multicast Backbone for Wireless Ad Hoc Networks". *In Proc. of the IEEE Intl. Conf. on Mobile Ad-hoc and Sensor Systems*, pp. 25-27.

[20] S. Srivastava & R. K. Ghosh, (2002) "Cluster based Routing using a k-tree Core Backbone for Mobile Ad hoc Networks". *In Proc. of the 6th Int. Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*. pp. 14-23.

[21] L. Xu, Z. Lin, D. Wang & J. Gao, (2006) "A Coloring Based Backbone Construction Algorithm in Wireless Ad Hoc Network". *In Proc. of the Advances in Grid and Pervasive Computing*, In Lecture Notes in Computer Science 3947, 2006, pp. 509-516.

[22] C. E. Perkins, E. M. Belding-Royer & S. Das, (2003) "Ad Hoc On Demand Distance Vector (AODV) Routing", RFC 3561.

[23] D. Johnson, D. Maltz, & J. Broch, (2001) "DSR: The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks", *Ad Hoc Networking*, Addison-Wesley, pp. 139-172.

[24] C. Perkins & P. Bhagwat, (1996) "Routing over Multi-hop Wireless Network of Mobile Computers". The Springer International Series in Engineering and Computer Science, *Mobile Computing*, 1996, 353: 183-205.

[25] The Network Simulator NS-2, http://www.isi.edu/nsnam/ns/.

[26] J. Broch, D. A. Maltz, D. B. Johnson, Y. C. Hu, & J. Jetcheva, (1998) "A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols". *In Proc. of the 4th ACM/IEEE Intl. Conf. on Mobile Computing and Networking*, pp. 85-97.

[27] M. Singhal & D. A. Manivannan, (1997) " Distributed Mutual Exclusion Algorithm for Mobile Computing Environments", *In Proc. of the IASTED International Conference on Intelligent Information Systems*.

[28] E. Gafni & D. Bertsekas, (1981) "Distributed Algorithms for Generating Loop Free Routes in Networks with Frequently Changing Topology", *IEEE Transactions. on Communications*, Vol. 29, pp. 11-18.

[29] Y. Chang, M. Singhal & M. Liu, (1990) "A Fault Tolerant Algorithm for Distributed Mutual Exclusion". *In Proc. of the 9th IEEE Symposium on Reliable Distributed Systems*, 1990, pp. 146-154.

[30] D. M. Dhamdhere & S. S. Kulkarni, (1994) "A Token Based k-Resilient Mutual Exclusion Algorithm for Distributed Systems". *IEEE Trans. on Communications*, Vol. 50, pp. 151-157.

[31] R. Baldoni, A. Virgillito & R. Petrassi, (2002) "A Distributed Mutual Exclusion Algorithm for Mobile Ad-Hoc Networks". *In Proc. of the Seventh IEEE Symposium on Computers and Communications*, pp. 539-544.

[32] M. Raynal, (1991) "A Distributed Solution for the k-out of-m Resources Allocation Problem", *In Lecture Notes in Computer Sciences*, Springer Verlag 497, pp. 599-609.

[33] J. R. Jiang, (2003) "A Prioritized h-out of-k Mutual Exclusion Algorithm with Maximum Degree of Concurrency for Mobile Ad Hoc Networks and Distributed systems". *In Proc. of the Fourth International Conference on Parallel and Distributed Computing*, *Applications and Technologies*, pp. 329-334.

[34] C.-Z. Yang, (2005) "A Token-based *h*-out of-*k* Distributed Mutual Exclusion Algorithm for Mobile Ad Hoc Networks". *In Proc. of the International Conference on Information Technology: Research and Education*, pp. 73-77.

[35] W. Wu, J. Cao & J. Yang, (2005) "A Scalable Mutual Exclusion Algorithm for Mobile Ad Hoc Networks", I*n Proc. of the International Conference on Computer Communications and Networks*, pp. 165-170.

[36] K. Erciyes, (2004) "Distributed Mutual Exclusion Algorithms on a Ring of Clusters". *In Lecture Notes in Computer Sciences*, Springer-Verlag 3045, pp. 518-527.

[37] O. Dagdeviren & K. Erciyes, (2007) "A Software Architecture for Shared Resource Management in Mobile ad hoc Networks". *In Lecture Notes in Computer Sciences*, Springer-Verlag 4362, pp. 224-234.

[38] S. Basagni, I. Chlamtac & V. R. Syrotiuk, (1998) "A Distance Routing Effect Algorithm for Mobility (DREAM)". *In Proc. of the 4th ACM/IEEE Intl. Conf. on Mobile Computing and Networking*. pp. 76-84.

[39] R. Casteneda & S. R. Das, (1999) "Query Localization Techniques for on Demand Routing Protocols in Ad Hoc Networks". *In Proc. of 5th ACM/IEEE Intl. Conf. on Mobile Computing and Networking*. pp. 186-194.

[40] P. Johansson, T. Larsson, N. Hedman, B. Mielczarek & M. Degermark, (1999) "Scenario-based Performance Analysis of Routing Protocols for Mobile Ad-Hoc Networks". *In Proc. Of the 5th ACM/IEEE Intl. Conf. on Mobile Computing and Networking*, pp. 195-206.

[41] Y. B. Ko, & V. H. Vaidya, (1998) "Location-aided Routing (LAR) in Mobile Ad Hoc Networks". *In Proc. of the 4th ACM/IEEE Intl. Conf. on Mobile Computing and Networking*, pp. 66-75.
[42] E. M. Royer & C. E. Perkins, (1999) "Multicast Operation of the Ad-Hoc on Demand Distance Vector Routing Protocol". *In Proc. of the 5th ACM/IEEE Intl. Conf. on Mobile Computing and Networking*. pp. 207-218.
[43] E. J.-H. Chang & R. Roberts, (1979) "An Improved Algorithm for Decentralized Extrema Finding in Circular Arrangements of Processes", *Communications of ACM*, Vol. 22, pp. 281-283.
[44] O. Dagdeviren, & K. Erciyes, (2008) "A Hierarchical Leader Election Protocol for Mobile Ad Hoc Networks". In Proceedings of the 8th international Conference on Computational Science, *In Lecture Notes in Computer Science, Springer-Verlag 5101*, pp. 509-518.
[45] R. U. Payli, K. Erciyes, & O. Dagdeviren, (2011), Cluster-based Load Balancing Algorithms for Grids, *International Journal of Computer Networks & Communications*, AIRCCSE, Vol. 3(5).
[46] S.Ayyasamy & S.N. Sivanandam, (2010) A Cluster Based Replication Architecture for Load Balancing in Peer-to-Peer Content Distribution, *International Journal of Computer Networks & Communications*, AIRCCSE, Vol. 2(5).
[47] Md. G. R. Alam, C. Biswas, N. Nower & M. S. A. Khan, (2012), A Reliable Semi-Distributed Load Balancing Architecture of Heterogeneous Wireless Networks, *International Journal of Computer Networks & Communications*, AIRCCSE, Vol. 4(1).

Authors

Kayhan Erciyes

Kayhan Erciyes received a BSc. degree in Electrical Eng. and Electronics from the University of Manchester, MSc. degree in Electronic Control Eng. from the University of Salford and a Ph.D. degree in Computer Engineering from Ege (Aegean) University. He was a visiting scholar at Edinburgh University Computer Science Dept. during his Ph.D. studies. Dr. Erciyes worked as visiting and tenure track faculty at Oregon State University, University of California Davis and California State University San Marcos, all in the U.S.A.

He also worked in the research and development departments of Alcatel Turkey, Alcatel Portugal and Alcatel SEL of Germany. His research interests are broadly in parallel and distributed systems and computer networks. More precisely, he works on distributed algorithms for synchronization in mobile ad hoc networks, wireless sensor networks and the Grid. Dr. Erciyes is the rector of the Izmir University in Izmir, Turkey.

Orhan Dagdeviren

Orhan Dagdeviren received the BSc. degree in Computer Eng. and MSc. degree in Computer Eng. from Izmir Institute of Technology. He received Ph.D. degree from Ege University, International Computing Institute. He is an assistant professor in International Computing Institute in Ege University. His interests lie in the computer networking and distributed systems areas. His recent focus is on graph theoric middleware protocol design for wireless sensor networks, mobile ad hoc networks and grid computing.