Arbitration-based Reliable Distributed Mutual Exclusion for Mobile Ad-hoc Networks

Murali Parameswaran
Department of Computer Science & Information Systems
Birla Institute of Technology and Science - Pilani,
Pilani Campus, Pilani, India 333031
Email: muralip@pilani.bits-pilani.ac.in

Chittaranjan Hota
Department of Computer Science & Information Systems
Birla Institute of Technology and Science - Pilani,
Hyderabad Campus, Hyderabad, India, 500078
Email: hota@hyderabad.bits-pilani.ac.in

***Abstract*—Distributed mutual exclusion enables critical resources to be shared amongst different mobile nodes in a Mobile Ad-hoc environment. In this paper, we place all nodes in a vicinity into regions. By suitably manipulating the behavior of arbitrator nodes, that form the bridge between two neighboring regions, we have ensured that permission is granted by every participating node, irrespective of the size of the network. We have used a single additional message, the $HOLD$ message, to ensure that the DME correctness is achieved for both inter-region and intra-region communications. Fault tolerance arguments for the proposed algorithm are also presented. To our knowledge, this is the first distributed mutual exclusion algorithm that uses the notion of regions and fault tolerance in MANETs.**

I. Introduction

Nodes deployed in a Mobile Ad-hoc Network (MANET) face frequent transient link failures and resource constraints due to the nature of the dynamically changing environment. Each mobile node's communication is limited to its neighboring set of nodes within their transmission range, which in turn are responsible for ensuring that the requisite packets are forwarded to the rest of the network. Communication links in MANETs are prone to frequent disconnections by virtue of distance or power restrictions.

Resource allocation is one of the challenging problems in such a dynamically changing environment. For the mutual exclusion problem, we need to guarantee that no two operations in critical section(CS) are concurrent. Apart from the usual requirement of safety and absence of deadlocks, any solution for the mutual exclusion problem must ensure that the fairness property is satisfied. By fairness, it is meant that any task wishing to enter CS shall not wait indefinitely to enter into CS. A centralized approach is inapproapriate due to the restrictions implied by the underlying network. In a dynamically changing environment, we need a distributed approach to solve the mutual exclusion problem.

Depending on the manner in which mutual exclusion is guaranteed, the solutions to the Distributed Mutual Exclusion (DME) problem can be classified into two broad categories: token-based algorithms and permission-based algorithms [1]–[6]. In short, a token-based approach [7]–[9] relies on the possession of a token to enter the CS. The token-based algorithms may be further classified into the circulating token method and the requesting token method. In the first method, a token is passed among all the participating nodes in a logical sequence. A node may enter CS only when it receives the token. After it completes its CS, the node will forward the token to its successor in the logical structure irrespective of whether the successor is the next node requested for entering CS or not. In the requesting token method, the onus of retrieving the token from the current holder falls on the node requesting to enter the CS. A permission-based algorithm relies on getting permission from all the participating nodes by explicitly asking for permission to enter the CS from each of the participating nodes. In this paper, we use the permission based approach for guaranteeing mutual exclusion.

While executing in CS, there is a possibility of unannounced death and process malfunction [10]. Apart from these two standard faults that can be expected in any environment, we need to consider transient loss of communication and probability for a node to abort itself (due to its residual energy dropping below a particular threshold) in the case of MANETs. It is safe to assume that a node will know a priori when it is forced to abort an execution due to power constraints. However, a transient failure due to loss of communication link to its neighboring nodes may not be predictable. Any such fault during the execution in CS may lead to a scenario where the other processes indefinitely wait for the faulty node to come out of the CS. Our algorithm guarantees reliability in such a mobile environment.

In a mobile environment, there is one more constraint that must be satisfied: combinatorial stability. While a node is waiting to decide whether it can enter CS or not, the underlying topology of the network may change due to high churn in the network. The running time of the algorithm must be fast enough to guarantee that the change in network state does not affect the current objectives of the algorithm. The notion of combinatorial stability is especially important for algorithms like token-based DME algorithms ([8], [9]) that contain an initiation system involving an elaborate set up of the underlying architecture and some permission-based algorithms([11]) that use clustering. The trick for ensuring combinatorial stability is to make local state information consistent, while ensuring the changes in global state can easily be updated. By making arbitrator nodes handle the request granting decisions locally, our algorithm provides combinatorial stability.

In this work, we have focused on permission based approach for the DME problem. Within the gamut of permission-based algorithms, we are using arbitration as the basis for handling DME among multiple regions. An arbitrator node that is aware of more than one region can easily look at the two regions and judge which node has the right to enter its CS as is being done by Maekawa in [6]. By using regions spanning multiple hops in diameter, the task of granting permissions among the nodes is better distributed by leveraging the available geographic information. We use arbitrators to decide which of the nodes in neighboring regions can be granted to enter CS without requiring a requesting node to ask for permission from a far away node. Our main contribution in this paper is the use of arbitration among multiple regions and providing reliability using appropriate fault-tolerance within the algorithm.

In Section II, we explore the previously proposed permission-based algorithms. Our algorithm is explained in Section III. Section IV presents the correctness argument for our algorithm. Relative performance is indicated in Section V. Finally, conclusion is outlined in section VI.

II. Literature Survey

A. Classical Algorithms

One of the earlier permission-based algorithms that could be extended for use in MANET was suggested by Ricart-Agrawala in [4]. In their algorithm, Ricart-Agrawala use only two messages $REQUEST$ and $REPLY$ for handling mutual exclusion. Here, the $REPLY$ message is used to indicate that a site has been granted permission to enter CS. A $REPLY$ message from the site in CS is equivalent to sending release messages to all the nodes. This algorithm has a message complexity of $2(N-1)$ messages, where N is the number of sites. The current modifications of this algorithm are discussed in the next sub section.

Maekawa [6] proposed a quorum-based distributed mutual exclusion algorithm that has a message complexity of $O(\sqrt{N})$. In this algorithm, the nodes participating in the distributed system are grouped together into quorums. The set of nodes in these quorums are designed in such a way that the intersection of any two sets is non-null. Thus a node needs to $REQUEST$ only to all the nodes in its quorum. Just communicating to these nodes directly is sufficient for finding out whether any other node is also requesting for CS. Each of the nodes to which $REQUEST$ is sent will verify that all nodes in the quorums it is participating are not requesting for CS. When it comes to MANETs, because of the characteristics of the underlying network, the additional overhead involved in the creation of quorums will weigh too much, compared to the benefits that can be reaped. The restrictions on creating the quorums typically limits the use of this algorithm in MANETs. As far as we know, there is no known algorithm that uses or mimics this approach in MANETs.

In [2], Singhal, *et al.* observed that a site need not consult other sites that are not currently contending for CS. In fact, the number of sites who are currently participating in the CS(Φ) is going to be smaller than N in most cases. They introduced the look-ahead technique to further reduce the message complexity to $2(\Phi-1)$. They used Dynamic Information Sets, comprising of Info_set and Status_set, to keep track of sites that are currently involved in CS.

B. Algorithms for MANETs

Wu, *et al.* in [3] extended Singhal, *et al.*'s algorithm to accommodate mobile nodes. They introduced three additional messages namely, $DOZE$, $DISCONNECT$ and $RECONNECT$. $DOZE$ message is used to indicate that a mobile node is going to a sleep mode to save power. $DISCONNECT$ and $RECONNECT$ messages were used to allow a mobile node to move out of communication range and return to the range respectively. To deal with fault-tolerance, they suggested the use of a timeout vector for $REQUEST$ messages, TO_{REQ}, which is maintained for the $REQUEST$ messages sent from the requesting site. If the TO^{i}_{REQ} expires, then the $REQUEST$ message is resent to the site S_i. This algorithm provides fault-tolerance by adding message types, and by providing time-outs. The other contribution of this paper is to provide an initialization mechanism for Singhal, *et al.*'s Dynamic Information Set in MANETs.

Parameswaran,*et al.* in [12] extended the algorithms proposed by Singhal, *et al.*, and Wu *et al.* to introduce fault tolerance and adaptive timeout handling mechanism. The $HOLD$ message proposed in our previous work enabled identification of nodes spending too much time in CS and failure of node in CS. The additional benefit of the $HOLD$ message is its ability to inform the requesting site about expected waiting time for entering CS, improving the reliability of the algorithm. This algorithm thus allows for variable execution time in CS.

Erciyes [5] had attempted in mapping the Ricart-Agrawala algorithm for MANETs. Instead of directly using the Ricart-Agrawala algorithm on all the nodes of the MANET, the $RA_{R}ing$ algorithm divides nodes into a set of logical coordinators that form a ring of nodes. These coordinators then use a modified Ricart-Agrawala technique to deal with the DME problem. Here, the message complexity is $O(k+3)$, where k is the number of coordinators. Synchronization delay varies from $2T$ to $(k+1)T$, which appears to be large when the number of clusters to be formed(k) becomes large. Erciyes, *et al.* [13] have used a weighted partitioning scheme described in [14] for creating clusters. However, the algorithm in [14] expects the number of partitions to be created as a parameter, which is not tenable in practical applications. In the process of trying to extend the Ricart-Agrawala algorithm with the help of clustering, unfortunately, variants of the $RELEASE$ message had to be brought back both in intra-cluster and inter-cluster communication. The use of rings for handing over permission to other cluster-heads also increased synchronization delay for this algorithm. Combinatorial stability is not guaranteed in this algorithm, especially under high mobility.

Gupta, *et al.* [11] suggested a cluster-based DME algorithm based on consensus based voting scheme. The number of votes for a cluster-head depends on the number of nodes within its cluster. The cluster heads coordinate among one another by using a version of representative voting. Each cluster head keeps track of whether it has allowed for a node to enter CS or not. When a $REQUEST$ is received, the cluster-head checks whether it has sufficient votes to let a node enter CS. If it doesn't have sufficient votes, then it asks for the votes of other cluster-heads. The cluster-heads maintain a $REQUEST$ queue to keep track of the oldest $REQUEST$ received by it to ensure fairness. This algorithm uses two release messages, as is used in [13]. This algorithm doesn't have combinatorial stability.

The current set of approaches for permission based algorithms thus focus on reducing message complexity either by reducing the effective number of participating nodes or by resorting to clustering. The clustering solution, however, would still perform worse than the classic Maekawa's algorithm due to the fact that communication overhead between cluster-heads is still relatively higher, especially in the case of large networks in which the participating nodes are spread out. While trying to divide the current node into clusters, the current set of algorithms are using only one-hop neighborhood for cluster creation. The cluster's periphery nodes transmit many of the messages, but they never make any decisions. This leads to overburdening of the cluster-heads. Since such hierarchical approaches look at inter-cluster and intra-cluster communication as two different sub-problems, they resort to a distributed version for handling DME among clusters, while resorting to a centralized algorithm with the cluster-head acting as the lone decision-making node within the cluster. There is a high computational as well as communication load on the cluster-heads, as they have to handle both cluster management and DME algorithm. Combinatorial stability is another factor that is not guaranteed by these algorithms. In our proposed algorithm, we relax the roles assigned to the nodes for region management and handling DME, thereby distributing the load to nodes in the periphery of the region as well.

III. Arbitration-based Reliable Distributed Mutual Exclusion Algorithm

In this section, we first discuss the general motivation, assumptions and the working of arbitrator. The data structures used in the algorithm are introduced. We discuss the algorithm along with pseudo-code of relevant routines at the end of this section.

A. Introduction and Motivation

The primary motivation for our algorithm is to ensure that every node should transmit the lowest number of effective messages in the underlying network, i.e., we are interested in reducing the communication delay for getting all the requisite permissions while still providing reliability and fault tolerance. If all the interested nodes are participating in CS, then this cannot be done. However, if we are able to exclude entire regions which are not currently participating, then we could reduce the effective synchronization delay.

One of the basic concepts for assigning quorums involves ensuring that for subsets S_i and S_j, $i \neq j$, $S_i \cap S_j \neq \emptyset$. Here, the intention is to ensure that every node in the subset becomes an arbitrator with some other subset. The performance improvement in Maekawa's algorithm [6] is derived from the fact that $N = K(K-1)+1$, where K is the size of subsets. In our algorithm, instead of trying to achieve the perfect distribution of nodes into subsets as required by [6], we relax the constraint without affecting much of the underlying concepts that drove the performance improvement.

In the proposed algorithm, we create subsets of nodes, R_i, such that all nodes that are members of R_i are within the same vicinity. An arbitrator, a for two regions R_i and R_j is chosen such that $a \in R_i \cap R_j$. We expand the notion of $HOLD$ message, suggested by Parameswaran,*et al.* in [12], to indicate that the current node itself, or some other node known to the current node, is either in CS or has a higher priority than the requester.

B. Assumptions

In this paper, we consider a distributed system consisting of N sites $(S_0 \ldots S_{N-1})$ participating concurrently for a shared resource in a mobile ad-hoc environment. We assume that each site contains only one process that is involved in the DME for the sake of convenience. This assumption is only being used so that we can refer to the participating processes with their site names instead of referring to the exact process number within the site. If there are more than one process within a site participating for CS, then we can always assign virtual site names for them.

In our work, both failure of a node and failure of the process which is running the code containing CS are considered one and the same, as the actions to be taken to continue execution are the same. The crashed or faulty node will take necessary steps to recover the data structures of the process, either by resetting the values or by using an older consistent set of values as has been suggested by Masum,*et al.* in [15]. The changes made shall be committed to the shared resource only just before exiting the CS. We assume that a site restarting after a crash will continue to work as they were just before the node entered the CS.

We assume that appropriate mechanisms are included in the nodes for handling network partitioning as is being done in [12] and [15].

We assume that the regions have been created a priori. For a simple region creation mechanism, we could use a two-level Maximal Independent Set (MIS) as can be derived from Alzoubi,*et al.*'s algorithm in [16], with a region radius of approximately 5 hops. We assume that there is also an appropriate region maintenance algorithm available for maintaining the network in the event of node mobility or node failure.

Since in [2] and [3], the problems of initializing the nodes are already dealt with; we are not going to discuss the node initialization strategies. For brevity of this paper, we assume that the region maintenance algorithm will take care of initialization of the Info_set and Status_set appropriately.

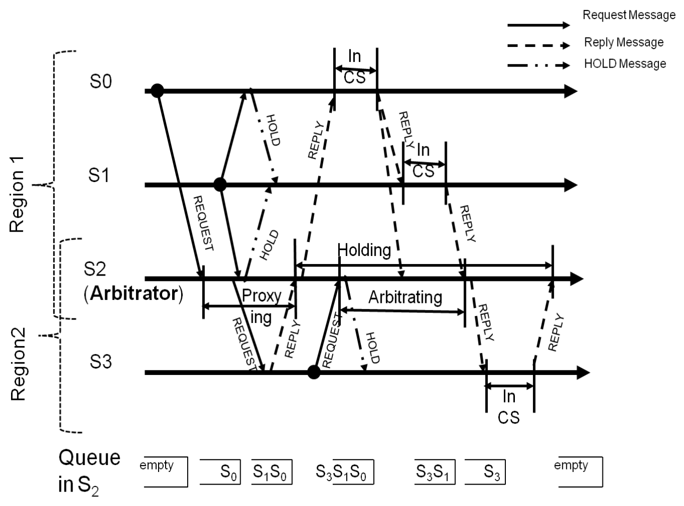

Fig. 1. Working of an Arbitrator.

The initialization of the relevant data structures is a trivial assignment of internal lists generated as part of two-level MIS creation. For our work, the only requirement is that the region maintenance algorithm can update the relevant data structures within the maximum intra-region communication delay.

Apart from the assumptions made by Wu,*et al.* [3] and Parameswaran,*et al.* [12], we assume that every participating node in CS is aware of the location of every other participating node in the network. This is a fair assumption as most of the current nodes do come with in-built GPS system just like the nodes using geographic routing algorithm.

We do not place any restrictions on the size of code fragment pertaining to the CS. We also assume that all sites requesting for CS, can continue their execution only after going to CS. Also, a site in CS will not remain there forever, but will come out of CS within a finite time. We also assume that the nodes are capable of differentiating between the actual sender of a particular message and the message initiator.

C. Arbitrator

The overall functioning of an arbitrator is illustrated in Fig.1. Here, S_2 is an arbitrator node that is common to region 1 and region 2. The site S_0, belonging to region 1, initiates a request for entering CS. As S_2 is aware that it is an arbitrator and there are no current requests pending with it, it will act as a proxy for S_0 in region 2, and send request message to nodes in its Info_Set in region 2. In the meantime, S_1 in region 1 also requests for entering CS. As S_0 and S_1 are both from the same region and S_1's priority is less than that of S_0, S_2 sends a reply message to S_1, while making an entry in its queue (listed in the bottom of the figure) for S_1. When S_0 gets S_1's request, it responds with a $HOLD$ message to inform S_1 that it is having a higher priority for entering CS. As soon as S_2 has received requisite permissions from region 2, it sends $REPLY$ to S_0 and moves from $proxying$ state to $holding$ state. Later, when S_3's $REQUEST$ message arrives in S_2, S_2 knows that it is currently holding for the nodes in its queue , it sends a $HOLD$ message back to S_3. As S_2 is now dealing with two (or more) regions, it will also mark itself as $arbitrating$. When the $REPLY$ message of S_0 reaches S_1, it enters CS. When the $REPLY$ message of S_0 reaches S_2, it identifies that there is still one more node in region 1 having a higher priority than S_3 and hence waits for the reply message from S_1. After S_1 has exited from CS, it will send reply message to S_2. When the $REPLY$ message of S_1 reaches S_2, it identifies that only nodes in one region are associated with it. S_2 resets the $arbitrating$ flag and grants permission for S_3 enter the CS. After S_2 gets the $REPLY$ message from S_3, it will come out of $holding$ state to indicate that all nodes in all regions associated with it are neither requesting to enter CS nor executing in CS.

D. Data Structures Used

Apart from the data structures used for the look-ahead technique described in Wu, *et al.* [3], we use the following data structures.

Reg: This list keeps track the different regions that the site belongs to, as well as the full list of all nodes in each of the associated regions. This list is mainly used by arbitrator nodes.

$Info\_set$: This set is the inform set defined by Singhal,*et al.* in [2]. Apart from the usual inclusions in this set, the list of arbitrators will also be added to this set. When a new arbitrator node is established, it will get added into this set.

$Status\_set$: This set is the status set defined by Singhal,*et al.* in [2]. Apart from the usual inclusions in this set, whenever the failure of a node in this set is identified, it will get automatically dealt with as if it is a non-participating node.

Q_{REQ}: This queue is used to maintain the list of nodes to whom we had sent request for the sake of fault-tolerance. When there is a change of arbitrator due to arbitrator node failure, the entry of the failed node will be purged from this queue.

TO_{CS_DONE}: A timeout meant to keep track of time left for the current node to exit its CS. This is the estimated time that the current node will be informing via the $HOLD$ message.

TO_{REQ}: The vector TO_{REQ} is used to maintain time-out values to sites to which the current node had sent a $REQUEST$ message.

TO_{HOLD}: A timeout vector to keep track of the timeout values sent by sites from which HOLD messages were received.

T_{CS_DONE}: This variable is used to set the initial value for TO_{CS_DONE}. The default value for this variable is the average time, τ, needed for CS as estimated by the process.

Q_{HOLD}: This priority queue is used by the site in CS and arbitrators to keep track of the set of nodes to which it has to send $HOLD$ messages.

$proxying$: This flag is used by an arbitrator to indicate that it is working on behalf of another node for getting permissions from other regions.

$holding$: This flag is used to identify whether the current node is aware that some other node is in CS. It is used primarily by arbitrators.

arbitrating: This flag is used by an arbitrator to indicate that it is currently acting on behalf of one region to another region. This flag is set only to enable an arbitrator site to easily identify whether it is dealing with multiple regions.

E. Our Algorithm

We only describe the modifications done to [12] to support the notion of arbitrators in this section. The handling of $DISCONNECT$ and $RECONNECT$ messages are not elaborated here, as their implementation is same as those by Parameswaran, *et al.* [12] and Wu, *et al.* [3]. The modifications of the dynamic information sets are done as indicated by Singhal,*et al.* in [2].

1) Requesting for CS: As in [12], when a site S_i wants to enter CS, it will send a $REQUEST$ message to all the nodes in $Info\_set_i$. The requesting site is either sent with a $REPLY$ message or a $HOLD$ message in response to its $REQUEST$. For every site S_j to which S_i has sent the $REQUEST$ message, it will set TO^j_{REQ} as the expected round-trip time between S_i and S_j.

2) Receiving $REQUEST$ message from site S_j: Algorithm 1 shows the steps taken by site S_i when it recieves a $REQUEST$ message from a site S_j. Apart from the procedure for dealing with $REQUEST$ message followed in [12], the site performs the following to take into account the multiple regions involved.

If the site S_i is an arbitrator, then it will mark the *holding* flag when it itself is in CS or when it has a higher priority than the requesting site S_j.

If S_i identifies that it is holding for one region while the request is coming from another region, it sets the *arbitrating* flag. If S_i is already arbitrating, then the site simply adds the incoming request to the Q_{HOLD}.

If S_i is not currently holding for some other node and it receives a request from S_j, then it identifies that it needs to act as a proxy for S_j. S_i transmits $REQUEST$ message to all nodes in the other region, making a note of the $REQUEST$ message in its request queue, just like a normal request initiated by it. While S_i is acting as a proxy, if it recieves further requests, it responds with a $HOLD$ message.

3) Receiving $HOLD$ message: Algorithm 2 shows the steps taken by the site receiving the $HOLD$ message.

When a site S_i receives a $HOLD$ message from a site S_j, it means that S_i's request could not be served because some other site is in CS or has a higher priority than itself. S_i acts the way it is done in [12]. If the site S_i is an arbitrator, it will check whether *proxying* flag is set. If so, then it marks the *holding* flag to indicate that some other site is in CS and puts the node in its request queue to Q_{HOLD}. It then sends a $HOLD$ message with updated estimated time inclusive of the time in the incoming $HOLD$ message to this site in Q_{HOLD}. If S_i has the *holding* flag set, it updates the holding timeout value, TO_{HOLD}, so that it can convey the new holding timeout value for the future.

Algorithm 1: Receiving $REQUEST$ Message from site S_j

```
begin
    if $S_i = S_j$ then
        Discard the message
        Exit procedure
    end
    if $InCS(S_i)$ then
        $Info\_set_i \leftarrow Info\_set_i \cup S_j$
        Send $HOLD(T_{CS\_DONE} - \delta^{expected}_{prop})$ to $S_j$
        $TO_{CS\_DONE} \leftarrow T_{CS\_DONE}$
        $Q_{HOLD} \leftarrow Q_{HOLD} \cup S_j$
        if $Arbitrator(S_i) = true$ then
            $holding_i \leftarrow true$
    else if $Requesting(S_i) \vee proxying_i$ then
        if $getPriority(S_i) > getPriority(S_j)$ then
            Send $HOLD(ts_{req})$ to $S_j$
            $Q_{HOLD} \leftarrow Q_{HOLD} \cup S_j$
            if $Arbitrator(S_i) = true$ then
                $holding_i \leftarrow true$
        else
            Send REPLY message to $S_j$
        end if
    else if $\neg InCS(S_i)$ then
        if $arbitrating_i$ then
            Send $HOLD(ts_{req})$ to $S_j$
            $Q_{HOLD} \leftarrow Q_{HOLD} \cup S_j$
        else if $holding_i$ then
            $arbitrating_i \leftarrow true$
            $Info\_set_i \leftarrow Info\_set_i \cup S_j$
            Send $HOLD(T_{CS\_DONE} - \delta^{expected}_{prop})$ to $S_j$
            Set $TO_{CS\_DONE} \leftarrow T_{CS\_DONE}$
            $Q_{HOLD} \leftarrow Q_{HOLD} \cup S_j$
        else if $Arbitrator(S_i)$ then
            $proxying_i \leftarrow true$
            foreach site $S_k \in Info\_set_i$ do
                Send $REQUEST$ message to $S_k$
        else
            Send $REPLY$ message to $S_j$
        end if
    end
    Wait till all sites have replied.
end
```

4) Receiving $REPLY$ message: When a site S_i receives a $REPLY$ message from a site S_j, apart from the way it is treated in [3], S_i checks the following in case it is an arbitrator. If *proxying* flag is set and the $REPLY$ received is the last $REPLY$ message it needs from that region, then it marks the *holding* flag and adds the site on behalf of which S_i had requested (stored in front of its request queue) to Q_{HOLD}. If the *holding* flag is set and S_i has received all requisite $REPLY$ messages (i.e. S_j was the last site not in its Q_{HOLD} to send the reply), it sends a $REPLY$ message to the node in front of Q_{HOLD}. We check *holding* flag separately and use $front(Q_{HOLD})$ to ensure that the site with the highest priority will get the right to continue.

Algorithm 2: Receiving $HOLD$ Message from site S_j with timeout value τ

```
begin
    $TO^j_{HOLD} \leftarrow max(\tau, TO^j_{HOLD})$
    if $proxying_i$ then
        $holding_i \leftarrow true$
        $S_{proxy} \leftarrow front(Q_{REQ})$
        $Q_{HOLD} \leftarrow Q_{HOLD} \cup S_{proxy}$
        Send $HOLD(TO_{HOLD} + \delta_{prop}(S_{proxy}))$ to $S_{proxy}$
    else if $holding_i$ then
        foreach site $S_k$ in $Q_{HOLD}$ do
            $TO^k_{HOLD} \leftarrow max(\tau + \delta_{prop}(S_k), TO^k_{HOLD})$
    end if
    Wait for $REPLY$ from $S_j$
end
```

IV. Correctness

A. Fault tolerance

Theorem 1: Based on assumptions, our algorithm effectively handles failure of both nodes and links.

Proof: For a single region, the fault tolerance arguments reduce to the proof for fault tolerance provided in [12]. Hence, proving that fault tolerance is ensured in the case of inter-region communication is sufficient to prove that our algorithm is fault-tolerant. In inter-region communication, we need to prove that link failure and node failure are correctly dealt with.

1) *Link failure between Sites S_i and S_j*: The proof argument is the same as that given in [12], as the behavior and assumptions are exactly the same two cases to look at.
2) *Node failure at site S_i*: Depending upon what S_i was maintaining, there are the following six cases to consider.

Case 1: S_i is not contending for CS and it is not an arbitrator.
If S_i is not contending for CS, then it is as good as it had disconnected from the network, and our algorithm handles accordingly.

Case 2: S_i is not contending for CS, but it is an arbitrator whose $proxying$ flag is set.

The failure of S_i will be treated as disconnection of the node from the network. The region maintenance algorithm will immediately assign some other node(/s) as the arbitrator(/s) connecting the regions affected. In the site S_j, on whose behalf S_i was acting as a proxy, the timeout TO_{REQ} will expire and will eventually be dealt with as a disconnected node. In the mean time, it would also observe that a new unrequested node has been included in its $Info\_set$. So, it sends a $REQUEST$ message to this newly created arbitrator. From this point on, the algorithm would continue with the normal mode of operation.

Case 3: S_i is not contending for CS, but it is an arbitrator whose holding flag is set.

This case again results in addition of new arbitrator(s), and the affected nodes' TO_{HOLD} or TO_{REQ} will expire, as the case may be, while new non-requested nodes will appear in their Info_sets. This will make the affected node(s) to send $REQUEST$ message to the newly created arbitrator(s) and the algorithm would continue with the normal mode of operation.

Case 4: S_i is contending for CS, but not in CS and it is not an arbitrator.

There are four situations to consider here.

- If a site S_j had send a $REQUEST$ message to S_i with a higher priority than S_i, and S_i had not replied; then the timeout mechanism will eventually inform S_j that S_i had disconnected and S_j will act accordingly.
- If a site S_j had send a $REQUEST$ message to S_i with a higher priority than S_i, and S_i had replied; then the failure of S_i is inconsequential to the working of the algorithm.
- If a site S_j had send a $REQUEST$ message to S_i with a lower priority than S_i, and S_i had responded with a $HOLD$ message; then the timeout TO_{HOLD} will eventually expire and S_j will know that the S_i had failed. S_j will now act as if S_i had disconnected and act accordingly.
- If a site S_j had send a $REQUEST$ message to S_i with a lower priority than S_i, and S_i had not responded with a $HOLD$ message; then the timeout TO_{REQ} will eventually expire and this treated as if S_i had disconnected.

Case 5: S_i is contending for CS, but not in CS and it is an arbitrator with Holding flag set.

This is handled in the same way as case 4, as the algorithm doesn't differentiate whether the incoming $REQUEST$ is from an arbitrator or not.

Case 6: S_i is in CS

If the site S_i was in CS when it failed, then in each site S_j in Q_{HOLD} the timer corresponding to TO^i_{HOLD} will expire. Following the mechanism for the expiry of TO_{HOLD}, $REQUEST$ message shall be resend and the sites will eventually discover that S_i had failed. This mechanism works exactly the same way whether S_i was an arbitrator or not. Following the identification of the failure of a node in CS, the appropriate mechanism for failure recovery of the application will kick in, as is assumed by our algorithm.

From the above six cases, it is clear that a failing node will be detected and handled correctly. Note that as $arbitrating$ flag is used only to keep track of whether S_i is having pending requests from more than one region, it is not going to affect the working of the algorithm in the event of node failure. ■

B. Mutual Exclusion

Lemma 1: Any site requesting for CS is aware of all the nodes, if any, waiting for CS with higher precedence than its own.

Proof: We follow the same argument as Lemma 1 in [12] for sites requesting within a region. We also need to consider the case of a $REQUEST$ sent to an arbitrator node. If an arbitrator node S_j receives a $REQUEST$ message from the site S_i, and it had responded with a $HOLD$ message, then the only reason why S_j had sent the $HOLD$ message is because it's $holding$ flag is set. This can happen only when S_j is aware of a site in CS or is aware of a site with a higher priority than S_i. Hence S_i becomes aware of another site in CS or with a higher precedence than itself in another region as well. ■

Theorem 2: Distributed mutual exclusion property holds for the algorithm.

Proof: Proof for a single region is trivial and is already done in [12]. Hence, we need to prove that mutual exclusion holds even in the event of simultaneous requests from multiple regions. The proof is by contradiction. Let S_i be the site with highest priority in the network that is contending for CS. Suppose there exist a site S_j, whose priority is lower than S_i, but still has entered CS, then it means that S_j has entered CS despite knowing that S_i has a higher priority than itself. From Lemma 1, every site S_j is aware of all sites S_i with a higher priority than itself. From Algorithm 1, all requesting sites with a lower priority would have recieved a $HOLD$ message from the site with the higher priority. Thus, a site will not enter CS in the presence of another site with a higher priority than itself as it would have recieved a $HOLD$ message and would be waiting for a $REPLY$ from the site with the higher priority as described in Algorithm 2. This is a contradiction. ■

C. Freedom from Deadlocks and Starvation

The proof for freedom from deadlocks is the same as that in theorem 3 of [12], as the modifications in the algorithm are not changing the dependence on receiving $REPLY$ messages for entering CS, and hence is not shown.

Theorem 3: The algorithm is free from starvation.

Proof: The proof is the same as that in theorem 4 of [12]. Though we have multiple regions, no site will be denied the right to enter critical section, if it has the highest priority in the network. From Lemma 1 and Algorithm 1, every node will allow all nodes with a higher precedence than itself to enter critical section, thereby allowing every node to enter its CS when its turn arrives. In the case of failure of an arbitrator node, once the system has stabilized after assigning new arbitrator(s) (as handled by the region maintenance algorithm), a node with a lower priority would still have to acquire permission from the arbitrator who will eventually become aware of the presence of a node with higher priority. ■

V. Performance

We performed comparative evaluation of the proposed DME algorithm, which we call as ARD, and algorithms by Erciyes, *et al.*(ED) in [13] and Gupta, (GRGK)*et al.* in [11] with the help of a simulator. All the three algorithms were tested by creating networks randomly for different number of nodes. While testing, we have deliberately inserted voids in the middle of the graph for half of the simulations to bring out worst case performance in our algorithm. A random subset of nodes were chosen to be participating nodes in CS. The entry time and duration of CS for each participating node was randomly chosen.

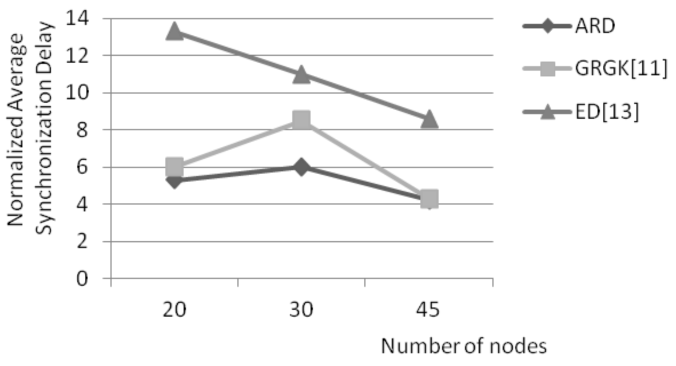

Fig. 2. Normalized Average Synchronization Delay v/s Number of nodes.

We have used the following three performance metrics for comparison. Normalized average synchronization delay is computed as the average synchronization delay per average communication delay for a single hop. The second parameter is the average number of hops per critical section. While these two metrics provide an intuitive feeling of the performance of a DME algorithm in most cases, we observed that a typical algorithm that spans multiple regions would suffer a little slack in performance for serving requests when no other node is contending for critical section. The normalized average time per critical section is a better measure for observing the performance of an algorithm over a large number of CS requests. This is computed as the average time needed for all the critical section operations to be completed along with the set of messages to bring it back to the original state. Unlike synchronization delay or response time, this metric considers the time needed for all messages related to the critical section, including the messages needed after completing CS.

Fig.2 shows the comparison based on normalized average synchronization delay between the three algorithms (ED [13], GRGK [11] and ARD). As can be observed, the three algorithms perform well with increase in the size of the networks with respect to synchronization delay. As the size of the network increases we observed that the excess delay in reaching to the next node requesting for CS via the intermediate node is less pronounced in ED and in ARD. GRGK were effectively sending the cluster release messages to the cluster head of the node with the next higher priority, once the requests were getting queued.

With respect to average hops per critical section shown in Fig.3, our algorithm seems to lag slightly behind ED ([13]). This is due to the extra set of packets that will get transmitted for maintaining fault tolerance. Our algorithm still performs much better than GRGK([11]).

In Fig.4, the normalized average time per critical section is shown. In GRGK([11]) and ED([13]), the node only needs to inform its cluster head after exiting from CS. In ARD,

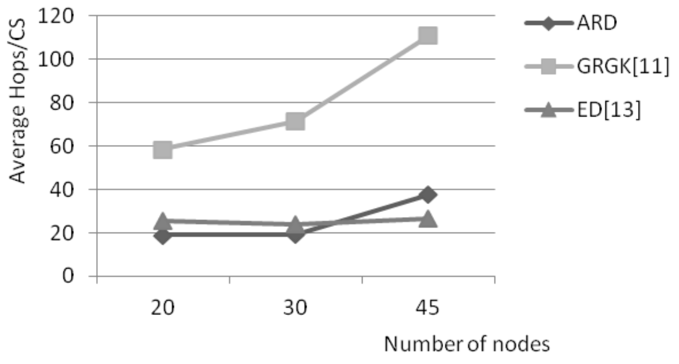

Fig. 3. Average Hops / Critical Section v/s Number of nodes.

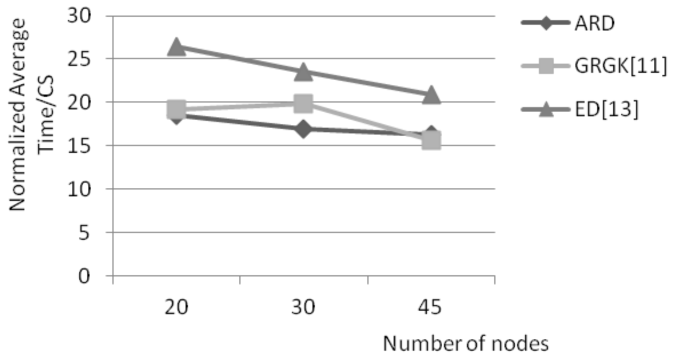

Fig. 4. Normalized Average Time per Critical Section v/s Number of nodes.

the the node that exits will eventually inform all affected arbitrators. But this has not affected the performance of the algorithm as compared to the other contemporary non-fault-tolerant algorithms. As can be observed, the time that a node would typically need to complete its CS is steadily reducing as the size of the network increases. Our algorithm performs much better than ED, while it is almost same as GRGK for larger size of the network. The slight degradation is due to the voids that we had introduced in our test cases.

VI. Conclusion

We have used $HOLD$ messages to handle DME across multiple regions as well as to provide fault-tolerance in our algorithm. Unlike other permission-based algorithms using multiple clusters, we use only three message types to handle the DME problem in a MANET spanning multiple regions. Our algorithm provides the same message complexity as [6] in the best case. Going forward, we would like to come up with more efficient region creation algorithms to augment our algorithm. Though we have assumed use of multicasting within a region, we would like to explore means of using only local flooding and unicast in conjunction with non-participating nodes to further reduce the number of actual packets that needs to be transmitted over the network. As far as we are aware of, this is also the first arbitration-based algorithm that has attempted to split the nodes into near-equal regions augmented with the look-ahead technique for MANETs. The algorithm proposed is the first fault-tolerant region-based distributed mutual exclusion algorithm for mobile ad-hoc networks.

References

[1] M. Benchaïba, A. Bouabdallah, N. Badache, and M. Ahmed-Nacer, "Distributed mutual exclusion algorithms in mobile ad hoc networks: an overview," *SIGOPS Oper. Syst. Rev.*, vol. 38, no. 1, pp. 74–89, Jan. 2004. [Online]. Available: http://doi.acm.org/10.1145/974104.974111

[2] M. Singhal and D. Manivannan, "A distributed mutual exclusion algorithm for mobile computing environments," in *Intelligent Information Systems, 1997. IIS '97. Proceedings*, 8-10 1997, pp. 557 –561.

[3] W. Wu, J. Cao, and J. Yang, "A fault tolerant mutual exclusion algorithm for mobile ad hoc networks," *Pervasive and Mobile Computing*, vol. 4, no. 1, pp. 139 – 160, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S157411920700051X

[4] G. Ricart and A. K. Agrawala, "An optimal algorithm for mutual exclusion in computer networks," *Commun. ACM*, vol. 24, no. 1, pp. 9–17, Jan. 1981. [Online]. Available: http://doi.acm.org/10.1145/358527.358537

[5] K. Erciyes, "Distributed mutual exclusion algorithms on a ring of clusters," in *Computational Science and Its Applications ICCSA 2004*, ser. Lecture Notes in Computer Science, A. Lagan, M. Gavrilova, V. Kumar, Y. Mun, C. Tan, and O. Gervasi, Eds. Springer Berlin Heidelberg, 2004, vol. 3045, pp. 518–527. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-24767-8_54

[6] M. Maekawa, "A n algorithm for mutual exclusion in decentralized systems," *ACM Trans. Comput. Syst.*, vol. 3, no. 2, pp. 145–159, May 1985. [Online]. Available: http://doi.acm.org/10.1145/214438.214445

[7] K. Raymond, "A tree-based algorithm for distributed mutual exclusion," *ACM Trans. Comput. Syst.*, vol. 7, no. 1, pp. 61–77, Jan. 1989. [Online]. Available: http://doi.acm.org/10.1145/58564.59295

[8] J. E. Walter, J. L. Welch, and N. H. Vaidya, "A mutual exclusion algorithm for ad hoc mobile networks," *Wirel. Netw.*, vol. 7, no. 6, pp. 585–600, Nov. 2001. [Online]. Available: http://dx.doi.org/10.1023/A:1012363200403

[9] Y. Chen and J. L. Welch, "Self-stabilizing mutual exclusion using tokens in mobile ad hoc networks," in *Proceedings of the 6th international workshop on Discrete algorithms and methods for mobile computing and communications*, ser. DIALM '02. New York, NY, USA: ACM, 2002, pp. 34–42. [Online]. Available: http://doi.acm.org/10.1145/570810.570815

[10] L. Lamport, ""the mutual exclusion problem: partii statement and solutions"," *J. ACM*, vol. 33, no. 2, pp. 327–348, Apr. 1986. [Online]. Available: http://doi.acm.org/10.1145/5383.5385

[11] A. Gupta, B. Reddy, U. Ghosh, and A. Khanna, "A permission-based clustering mutual exclusion algorithm for mobile ad-hoc networks," *"International Journal of Engineering Research and Applications"*, vol. 2, no. 4, pp. 019–026, Jul. 2012. [Online]. Available: http://ijera.com/papers/Vol2_issue4/D24019026.pdf

[12] M. Parameswaran and C. Hota, "A novel permission-based reliable distributed mutual exclusion algorithm for manets," in *Wireless And Optical Communications Networks (WOCN), 2010 Seventh International Conference On*, sept. 2010, pp. 1 –6.

[13] K. Erciyes and O. Dagdeviren, "A distributed mutual exclusion algorithm for mobile ad hoc networks," *"International Journal of Computer Networks and Communications"*, vol. 4, no. 2, pp. 129–148, Mar. 2012. [Online]. Available: http://www.airccse.org/journal/cnc/0312cnc09.pdf

[14] O. Dagdeviren, K. Erciyes, and D. Cokuslu, "Merging clustering algorithms in mobile ad hoc networks," in *Distributed Computing and Internet Technology*, ser. Lecture Notes in Computer Science, G. Chakraborty, Ed. Springer Berlin Heidelberg, 2005, vol. 3816, pp. 56–61. [Online]. Available: http://dx.doi.org/10.1007/11604655_9

[15] S. M. Masum, M. M. Akbar, A. A. Ali, and M. A. Rahman, "A consensus-based -exclusion algorithm for mobile ad hoc networks," *Ad Hoc Networks*, vol. 8, no. 1, pp. 30 – 45, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570870509000237

[16] K. M. Alzoubi, P.-J. Wan, and O. Frieder, "Message-optimal connected dominating sets in mobile ad hoc networks," in *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, ser. MobiHoc '02. New York, NY, USA: ACM, 2002, pp. 157–164. [Online]. Available: http://doi.acm.org/10.1145/513800.513820