Rapport de Recherche

Distributed Mutual Exclusion Algorithms In Mobile Ad Hoc Networks: An overview*

M. Benchaïba, A. Bouabdallah, N. Badache and M. Ahmed-Nacer

LSI-TR-0303         Juin 2003
Distributed Mutual Exclusion Algorithms In Mobile Ad Hoc Networks: An overview*

M. Benchaïba¹, A. Bouabdallah², N. Badache¹ and M. Ahmed-Nacer¹

¹ LSI, Dep. of Computer Science, USTHB, B.P. 32 El-Alia, Bab Ezzouar- Algiers – Algeria,
² Heudiasyc, UMR-CNRS 6599, UTC, B.P. 20529- 60205 Compiègne Cedex, France
mbenchaiba@wissal.dz, bouabdali@utc.fr, badache@wissal.dz, anacer@wissal.dz


Abstract

The problem of mutual exclusion has been extensively studied in distributed systems. The proposed solutions can be mainly classified in consensus based and token based protocols. Some of the proposed solutions consider the physical topology of the networks and try to provide optimal message exchange and minimal synchronisation delays. Others, impose a logical structure on the network like a ring or a tree. Recently, the mutual exclusion problem received an interest for mobile ad hoc networks. These networks are known as a challenging domain. To our knowledge, few algorithms have been proposed in the literature and all of them are token based approach. In this paper, we review the distributed mutual exclusion algorithms developed for mobile environments and principally for ad hoc networks and discuss some issues.

Keywords: Mobile Ad hoc Network, Algorithm, Mutual Exclusion, Critical Section.

1. Introduction

Wireless networks is an emerging new technology that allow users to access information and services regardless of their geographic position. Wireless networks can be classified in two types: infrastructured network and infrastructureless (ad hoc) networks. Mobile ad hoc networks are very useful in emergency search-and-rescue operations, meetings or conventions in which persons wish to quickly share information, and data acquisition operations in inhospitable terrain.
A mobile ad hoc network can be defined as a network that is deployable spontaneously and is independent from any predefined static network. The network is composed of a collection of mobile nodes with wireless interfaces and forms an arbitrary and dynamic topology. Each node is able to communicate with nodes in its transmission range and acts as processor and router to route messages. The network is characterised by limited bandwidth and energy. So, each node is subject to frequent disconnections and doze mode functions.

Resource allocation (mutual exclusion, assignment of channels[17], IP address [27], etc.) is one of the most important problem in these challenging networks. The network topology changes frequently due to mobility[36], disconnections, failures of nodes and network partitioning. So, the resource allocation solutions provided for fixed networks can not be applied to mobile ad hoc networks. So, the problem of sharing a critical resource in mobile ad hoc networks have received an interest during these recent years. Mutual exclusion solutions that have been proposed for fixed networks can be classified in two types: centralized approach in which a node is designated as coordinator to deliver permission to the other nodes to access their CS, and the distributed approach in which the permission is obtained from consensus between all network nodes. Due to the symmetry role of nodes and the characteristics of the networks, the first approach is not suitable for mobile ad hoc networks.

The distributed mutual exclusion algorithms are mainly classified in two categories: Permission based [1] [5] [6] [14] [21] [23] [31] [37] and token based [7] [8] [11] [24] [25] [26] [29] [30] [34]. Permission based mutual exclusion algorithms impose that a requesting node is required to receive permissions from other nodes (a set of nodes or all other nodes). In the token based algorithms, a unique token is shared among the set of nodes. The node holding the token is allowed to enter its critical section. The problem of mutual exclusion in mobile ad hoc networks is challenging and have received interest in recent years, but few solutions have been proposed.

In this paper, we focus on the distributed mutual exclusion problem in mobile ad hoc networks. We review the existing solutions and outline the limitations of each presented algorithm. The rest of the paper is organized as follows. In section 2, we present the mutual exclusion problem. Section 3 describes some mutual exclusion algorithms developed for fixed topology that were adapted for mobile ad hoc networks. Mutual exclusion protocols proposed in the literature for mobile ad hoc networks are presented in section 4. In section 5, we discuss briefly these algorithms and conclude the paper in section 6.

2. The mutual exclusion problem

Distributed mutual exclusion provide access to shared critical resources (resources which may be accessible by a single process at a time). Several solutions have been proposed in the literature for fixed networks. These solutions can be classified into two categories: the token based and permission based. With token-based solutions, there exists a unique token in the system and only the node holding the token may access the critical section. To access the critical section using permission-based solution, a process \( p_i \) is required to receive permission from a set of nodes \( S_i \). During the recent years, some mutual exclusion algorithms have been proposed for mobile ad hoc networks. The distributed system that we consider is composed by a set of \( N \) nodes communicating by message exchange. Every node may communicate directly with its neighbours by exchanging messages and keeps information about its neighbours or about all the
nodes. The communication delay is assumed to be finite but not bounded. The topology of the network is arbitrary, in mobile ad hoc networks the topology may change during the time. Any mutual exclusion algorithm have to ensure two properties: Safety and Liveness. The safety property ensures that at most one process is executing its critical section at any time, while the liveness property ensures that a requesting node will succeed to enter its critical section in a finite time. Performance of distributed mutual exclusion algorithms are evaluated by the number of messages generated per critical section entry, synchronisation delay, and size of information control.

3. Related works

In this section, we recall the main idea of distributed mutual exclusion and some algorithms developed in the literature for fixed networks on which recent mutual exclusion algorithms for mobile ad hoc networks are based.

In the token based approach, two methods are usually used circulating token and requesting token. In the requesting token method, a node requesting the CS has to obtain the token. The basic problem is how to reach the token holder. In some algorithms, the request is sent to all the nodes because the token holder is unknown[32], in others a logical structure is defined to point the token holder, for example a Direct Acyclic Graph[29]: The request is sent over the branch of the DAG which leads to the node holding the token.

In the solution proposed by Le Lann[20], all nodes are logically organized in an unidirectional ring and the token circulates following this ring. When the token is received by a node, it enters its CS if it is requesting, and after executing its CS, it sends the token to its successor in the ring.

In[32], Ricart and Agrawala proposed an algorithm which requires at most $N$ messages to achieve mutual exclusion. The requesting node sends the request message to all the other $N-1$ nodes and waits for responses. When the process which holds the token has to send the token, the next process is chosen in a circular manner if any, otherwise, it keeps the token in an idle state. Based on the Ricart-Agrawala’s algorithm, Suzuki and Kazami [37] proposed an algorithm in which the queue of requesting nodes is piggybacked within the token. This queue is updated by a local queue of each visited node in an ascending node number in order to ensure the liveness property. Singhal [35] has improved the performance of the Suzuki and Kazami algorithm, to at most $N$ messages in heavy loads. He used a heuristically method to guess what nodes of the system are probably holding or are likely to have the token and sends a token request message only to those nodes rather than to all the nodes. To achieve this, the knowledge of each node about the requesting nodes is passed through the token.

In[29], Raymond developed an algorithm, based on a logical tree on the network rooted by the token holder node, which requires at most $O(\log N)$ messages to enter the CS. The tree is maintained by the logical pointers distributed over the nodes and directed to the node holding the token. When a node wants to access its CS, it enqueues its identity and sends a request message to the next node in the direction of the token holder, this message is then routed successively to the token holder. The token is sent back over the reverse path to the requesting node. The direction of the link of the token sending nodes must be reversed so to point always the token.
holder. In [9], Chang, Singhal and Liu developed an algorithm which improves Raymond’s algorithm which tolerates link and node failures by maintaining multiple paths to search the token. The algorithm tries also to avoid cycles when the token returns to the requester along the reversal links. In [11], Dhamdhene and Kulkarni developed an algorithm which aims to resolve the problem of still remaining cycle in the Chang, Singhal and Liu algorithm, and it is k-resilient, that is it tolerates k node/link failures.

In permission based approach, to enter its CS, a requesting node must wait to receive responses from all the other nodes in the network (or from a set of nodes in some algorithms).

In the algorithm proposed by Lamport[19], when a node wants to enter its CS, it sends a request to all other nodes and waits for reply messages. When it exits its CS, it sends a release message to all other nodes. This algorithm requires \(3(N-1)\) per critical section entry.

In [31], Ricart and Agrawala proposed a distributed algorithm which requires \(2(N-1)\) messages per critical section entry. When a node wants to enter its CS, it sends a request to all the nodes of the network and waits for responses. If it receives an agreement of all these nodes, it enters its CS. Requests are ordered by Lamport clocks[19]. In [21], Maekawa has developed an algorithm which requires \(c(\sqrt{N})\) messages to enter the CS. The algorithm uses a logical structure[2] defined by a set of nodes associated with each node and this set has a non null intersection with every set associated to each node. This structure allows each node, which wants to access its CS, to have permission only from each member of the set associated to it.

4. Mutual exclusion in mobile ad hoc networks

4.1 Context

A Mobile ad hoc network is a network that is deployable spontaneously and is independent from any predefined static network. The network is composed by a set of mobile nodes with wireless interfaces and forms an arbitrary and dynamic topology. Each node is able to communicate with nodes in its transmission range and function as processor and/or router to vehicle messages to other nodes. The network is characterised by limited bandwidth and energy. So, each node is subject to frequent disconnections and doze mode functions. So, this requires that the algorithms developed for mobile ad hoc networks must be efficient in particular in terms of low energy consuming. The transmission delays of messages are variable since their proximity changes. The existing algorithms developed for fixed networks could not be applied in mobile ad hoc networks due to these properties.

In mobile ad hoc networks, most of the existing works have been provided at medium access level[18] and routing level[28][16] which ensures message delivery to any reachable node, but very few works have been done for distributed services and particularly in mutual exclusion. Classical distributed mutual exclusion algorithms may function in mobile ad hoc networks directly on top of routing protocols. In this case, these algorithms do not take care of the node mobility, consequently these algorithms may induce catastrophic performances. The second approach requires that mutual exclusion algorithms take into consideration the movement of nodes and then the topology changes.
The existing distributed mutual exclusion algorithms for mobile ad hoc networks are obtained from the adaptation of those developed for fixed networks in order to deal with nodes mobility. The performance analysis considers a set of metrics: *Message complexity*, *Synchronisation delay*, *Local storage*, and *messages size*. These evaluations could be done under two situations: heavy and light loads.

4.2 Algorithms in mobile ad hoc networks

In the following, we present existing distributed mutual exclusion algorithms developed for mobile ad hoc networks except the first one[3] which is defined for cellular networks.

4.2.1. B. R. Badrinath, Arup Acharya and T. Imielinski algorithm[3]

In [3], B. R. Badrinath, Arup Acharya and T. Imielinski proposed two distributed mutual exclusion algorithms for cellular networks. The first one is an adaptation for cellular networks of the algorithm proposed by Lamport[19], the second one is an adaptation for cellular networks of the algorithm proposed by Le Lann[20]. A cellular network is formed by two parts: a fixed wired part or a backbone and a mobile part in which every mobile node communicates by using a wireless interface. When a mobile host wants to communicate with another one, it sends a message to the *base station* to which it is attached, this base station forwards the message to the base station of the receiver mobile host, and finally the message is delivered to the destination. Mobile hosts have several resource constraints in terms of limited battery life and often operate in doze mode or entirely disconnect from the network. So, it is recommended to do most computation on the static part of the network, and the role of a host should be only to initiate a computation and to receive results.

The Lamport algorithm is adapted to mobile computing environments by shifting the communication and computation requirement of the algorithm to the static part of the network. i.e. mutual exclusion is achieved with better message complexity because the base station acts as a proxy for nodes attached to it. Each host is replaced by its base station which maintains necessary data structures. The base stations exchange timestamped requests, reply and release between them and ensure mutual exclusion on behalf the hosts. Each host is limited then to send its request to only its base station, receive the grant from only its base station, and sends a release message to its base station when it exits its critical section. If the host disconnects prior to receiving the grant request, the base station releases the resource to another host attached to it. If the host disconnects after receiving the grant request but without sending release resource, it must reconnect to send release resource.

The adaptation of Le Lann's algorithm to a mobile environment is made by arranging the base stations (instead of hosts) in a logical ring and each base holds a FIFO *queue* which contains the requests of hosts attached to it. The token circulates on the logical ring and when it is received by a base station, it is sent to the requesting hosts but to one at a time until the queue is empty (the queue maintained by the base station), and then it is passed to its successor in the
ring. To ensure that a host enters its CS at most once during one tour (and finally to ensure fairness), a counter of rounds is included in the token message. This adaptation has several advantages: It avoids to frequently communicate with hosts and finally reduce considerably the cost for the token to traverse the ring. Also, when a host disconnects, the logical ring has not to be reorganized. In the other hand, a host operating in a doze mode is interrupted only to satisfy its prior request.

4.2.2. J. Walter and S. Kini algorithm [40]

In [40], J. Walter and S. Kini proposed a token based mutual exclusion algorithm derived from several other algorithms: the tree based routing protocol of Gafni and Bertsekas [13], the algorithm presented in [9] and others ideas from [11][29]. This algorithm defines a structure mapped on real topology of the network which is represented by a Direct Acyclic Graph (DAG) of token-oriented pointers, maintaining multiple paths leading to the node holding the token. The algorithm is well suited to the distributed mobile setting because it requires nodes to keep information only about their immediate neighbours. In the absence of node’s movement, the algorithm acts as Raymond's algorithm[29] to forward requests to the node holding the token and sends back the token to the requesting node. Moreover, each node keeps an elevation information (see below) which is used to update the DAG structure in case of link failure in order to have always the token holder node as a root of this tree. This algorithm assumes that failures occur only due to node movement. It is assumed that the token cannot be lost and communication links are bidirectional. The nodes move with a limited speed so it can not disconnect from the network during activation of the algorithm and during the message transmission.

The algorithm starts by constructing a token-oriented connected DAG, maintained by the logical pointers distributed over the nodes and directed to the token holder. These pointers are defined by the relative elevation of a node in relation to its neighbouring nodes. The elevation of a node \(i\) has the form \((\alpha, \beta, i)\) and increase according to its distance from the token holder (see [13]). The neighbours of each node are divided in two sets: a set of nodes connected to incoming links and a set of those connected to outgoing links in order to maintain many routes to the node holding the token. Requests are forwarded to the token holder along the tree, and each node sends at most one common request message for both itself and its incoming neighbours (when it receives the first request from one of its neighbours). To do so, it maintains a request queue to store and order requests and also for backward path to reach the requesting node. Upon receiving a token, the node which detects its own id is in the top of requesting queue becomes a “sink” by modifying its elevation to be lower than its neighbours and finally enters the CS. The token holder will always be the lowest node in the DAG. So, the partial rearrangement of the DAG is necessary.

During the execution of the algorithm, some links may fail and/or may be created. When, the node holding the token exits its CS, it sends the token to the node which is in the top of the request queue if any, otherwise it maintains it locally. In the normal case, the token is delivered over the reverse path to the requesting node. If one or more links on the reverse path fail, a search for the next node is initiated by the token holder node by sending search messages on all
its incoming links. Each node receiving this message for the first time, propagates it similarly on all its incoming links. When the next node on the path is found, the token is forwarded again.

In figure 1, part (a) depicts a simple mobile ad hoc network. Part (b) shows the state of the network with logical links after initialisation of the algorithm. Node D is holding the token (has no outgoing links) and all other nodes point to node D. Along the paths to token holder node, the elevation decreases. Node A issues a request for the token over path A which is enqueued on F’s request queue, and forwards the request to node E which enqueues the F’s request. And finally D enqueues E’s request.

![Diagram](image)

**Figure 1:** Rearrangement of the tree after a link failure.

When a node detects a failure of its last existing outgoing link, this results that there is no path to the token holder. Then, it invokes a partial rearrangement of the DAG using the method described in [13] which avoids the formation of a cycle. Generally, the failure may happen at any time, the algorithm provides the mechanisms to ensure its normal functioning. In the example given in figure 1(c), the link C-D has failed due to an increase in the distance between nodes C and D. The failed link is the last outgoing C which does not hold the token. So, C rearranges its links, changes its elevation that causes node B to have no outgoing links. Also, after this, G has any outgoing links. Consequently, the tree is partially rearranged.

Figure 1(d) shows the result of the token movement from D to A causing the elevation changes of the nodes E, F and A. These changes ensure that all logical links point from nodes with higher elevation towards nodes with lower elevation.
When a new link is detected, the two adjacent nodes of this new link exchange messages to achieve the necessary modifications of outgoing and incoming links. This operation may cause the loss of the outgoing links of each receiving node, so, the partial rearrangement of the DAG is triggered.

The algorithm guarantees the safety and liveness property (see [40] for the proof). For complexity analysis, the proposed algorithm (which considers the node movements), called RL_L2, is compared with the one presented in [29] which is executed on top of the routing algorithm TORA[28], called TORA_L1. When the nodes are not mobile, the two protocols exhibit nearly the same performances \(O(p)\), \(p\) is the longest path in the DAG) in terms of messages exchanged because the logical routes maintained by the mutual exclusion protocol correspond to that maintained by the routing protocol. But, in time complexity, the first algorithm is better because the second uses a stack of protocols. When node mobility is considered, the first algorithm exhibits fewer messages and less time than the second because the last one is independent of the routing protocol used. Details are described in table 1[40]. The parameters considered are: the number of network nodes \(n\); the number of nodes in a network segment affected by a topology change \(l\); the longest directed path in an affected network segment \(p\), the maximum nodal degree \(\phi\), and the time for a message to be passed up the protocol stack to the application level \(\delta\).

<table>
<thead>
<tr>
<th>Requests</th>
<th>RL_L2</th>
<th>TORA_L1</th>
<th>RL_L2</th>
<th>TORA_L1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mobility</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No mobility</td>
<td>(O(p))</td>
<td>(O(p))</td>
<td>(O(p))</td>
<td>(O(\delta p))</td>
</tr>
<tr>
<td></td>
<td>(O(p))</td>
<td>(O(np))</td>
<td>(O(p))</td>
<td>(O(\delta np))</td>
</tr>
</tbody>
</table>

| Token Movements   |        |         |        |         |
| Mobility          |        |         |        |         |
| No mobility       | \(O(p)\) | \(O(p)\) | \(O(p)\) | \(O(\delta p)\) |
|                   | \(O(np)\) | \(O(np)\) | \(O(np)\) | \(O(\delta np)\) |

| Link Failures     |        |         |        |         |
| Mobility          |        |         |        |         |
| No mobility       | \(N/A\) | \(N/A\) | \(N/A\) | \(N/A\) |
|                   | \(O(l\phi)\) | \(O(l\phi)\) | \(O(l\phi)\) | \(O(\delta l\phi)\) |

Table 1: Complexity of RL_L2 and TORA_L1

We can mention that the proposed algorithm considers that mobility is slow. In another hand, it doesn’t deal with token loss and network partitioning and merging. For performance, a comparative study is made only by considering the same algorithm but on the top of TORA routing algorithm. So, more intensive simulations should be done in order to get a more precise idea on the performances of this algorithm.

4.2.3. J. Walter, J. Welch and Vaidya algorithm [41]

In[41], J. Walter, J. Welch and Vaidya proposed an alternative solution of the one described above[40] (see 4.2.2), but uses always as a basis the Raymond algorithm[29]. The same assumptions as the previous algorithm are done. In addition to that, it is assumed that communication channels are FIFO with no loss nor duplication of messages. The proposed algorithm uses a structure mapped on physical topology of the network which is represented by
a direct acyclic graph (DAG) of token-oriented pointers, maintaining multiple paths leading to
the node holding the token. Network nodes keep information only about their immediate
neighbours. Each node dynamically chooses its neighbour with lowest elevation as its preferred
next link to the token holder. When a link fails, the concerned node reroutes its request to another
path. All requests reaching the token holder are treated symmetrically.

Requests are forwarded to the token holder over the tree like in [40]. The token is delivered
over the reverse tree path to the requesting node. The token holder will always be the lowest
node in the DAG. Sometimes, links may fail and may be created. We note also that the main
difference with the algorithm presented in[40] is that in this algorithm when the process wants to
send a token, it finds necessary the return path (because no partitioning of the network). So, the
failure is treated by the requesting node (because each node is aware of its neighbouring nodes).
This eliminates the overhead introduced by the search process. Note that the partial
rearrangement of the DAG may be necessary when the token circulates or when links may be
created or failed. We note also that this algorithm uses less message types than in[40].

When a node detects the failure of an outgoing link and it is not the last outgoing one, it
reroutes the request. If it is the last outgoing link, there is no path to the token holder, so, it
invokes a partial rearrangement of the DAG to find a new route. When a new link is detected,
the two nodes concerned with this fact exchange messages to achieve the necessary changes in
their outgoing and incoming links and to reroute eventually their requests. So, the partial
rearrangement of the DAG is called. Liveness and safety properties are proved.

The simulation were performed by comparing the performance of this algorithm called RL with
the Raymond’s algorithm[29] from which RL is adapted. The Raymond’s algorithm (which is
called here RR) is executed on the top of a routing layer that always provides shortest path routes
between nodes. Two parameters are considered, the average waiting time for CS access and the
average number of messages per CS entry. For average waiting time, the results shows that the
RL algorithm is better than RR algorithm when nodes are mobile and not far in some others.
The RR algorithm exhibits also poor results when connectivity is low because it considers
logical path routes between nodes. For the average number of messages per CS entry, the RR
algorithm sends fewer messages per CS entry than the RL algorithm in all simulation trials.

4.2.4. R. Baldoni, A. Virgillito algorithm [4]

The algorithm proposed by R. Baldoni, A. Virgillito [4] is based on a dynamic logical ring
and combines the two methods token-asking and circulating token. The algorithm aims to
maintain device power consumption as low as possible by reducing the number of hops traversed
per CS execution and by avoiding to send any control message when no process requests 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. In this algorithm, authors assume that
processes do not crash and the network is reliable. The network is not subject to permanent
partitions. If a partition occurs, each pair of processes will be eventually able to communicate,
and finally, each process may query at each time the information provided by the routing
protocol.

The algorithm continuously executes transitions alternately between two states: Idle and
Coordinator-Change and is executed in rounds. For each round, that is materialized by
circulating a token over all nodes, one process is designated to be a coordinator (see below), say \( c_k \), to which processes have to send requests. The round is completed when all nodes of the network are visited. The round is aimed to inform each process of the ring on the new coordinator and to allow the receiving and requesting one to enter the CS.

When a process wants to access its CS, it sends a request to the coordinator and waits for the token. The coordinator inserts the received request in a `PendingRequest` queue ordered according to a policy \( p \) (say FIFO for simplicity). Initially, the algorithm is in an `idle` state and the coordinator \( c_0 \) is the process \( p_1 \) and all other processes know this fact. The coordinator \( c_k \) transits the system from `idle` state to `coordinator-change` state when the `pendingRequest` queue is not empty. The coordinator sends then the token to the first process in the queue, say \( p_j \), which becomes the coordinator \( c_{k+1} \). During the movement of the token along the ring (of \( n-1 \) processes and that does not contain the last coordinator \( c_k \)), each process receiving the token enters the CS if it is requesting. After this, the receiving process computes dynamically and on-the-fly the components which are not yet visited (by using the Boolean array `receivedToken` within the token which indicates the processes that have already received the token in the actual round), designates its successor if any, and sends him the token. Otherwise, it sends the token to the coordinator \( c_{k+1} \) (which changes its state to `Idle`). The successor is computed by using a policy, which provides a deterministic choice, based mainly on the distance (number of hops) between the sender and the receiver of the token. The disconnected processes are considered to be at infinite distance, and they are treated with waits and tries to transmit.

The algorithm is proved that it ensures safety because a process does not enter its CS until it receives the token and the token is sent to exactly one process in each time. The `liveness` property is ensured because one of the token structure is a vector which indicates the visited nodes in the current round and the round does not finish until every node in the network has been visited.

Performance is studied by computing three parameters: `message complexity`, `synchronisation delay` and the `size of control information` piggybacked with each message. These measurements are computed under two situations: `heavy` and `light load`. Table 2 shows results and comparison between the proposed algorithm and three others well-known token-asking ones: Suzuki-Kazami [37], Singhal [35] and Raymond [29] on the top of a routing protocol.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Sync. Delay (Heavy load)</th>
<th>Message (Heavy load)</th>
<th>Message (light load)</th>
<th>Control Information Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baldoni-Virgillito</td>
<td>T</td>
<td>2</td>
<td>N</td>
<td>O(n)</td>
</tr>
<tr>
<td>Suzuki-Kazami</td>
<td>T</td>
<td>N</td>
<td>N</td>
<td>O(n log n)</td>
</tr>
<tr>
<td>Singhal</td>
<td>T</td>
<td>n/2</td>
<td>N</td>
<td>O(n log n)</td>
</tr>
<tr>
<td>Raymond</td>
<td>T(log n)/2</td>
<td>4</td>
<td>Log n</td>
<td>O(1)</td>
</tr>
</tbody>
</table>

Table 1: Performance comparison of token-asking algorithms.

Simulation has been established with GloMoSim environment[42]. The results show that the hop counts per CS entry decrease when the load request increases accordingly to the theoretical
results, where mobility has no significant impact on this measure. In other hand, the number of hops by application message increases when mobility is lower.

4.2.5. N. Malpani, N. H. Vaidya, J. L. Welch algorithm [22]

In [22], N. Malpani, N. H. Vaidya, J. L. Welch proposed a parametric algorithm with many variants. It uses a logical dynamic/variable size ring (on which the token circulates): the size of the ring may vary at every round (in which all nodes must be visited). All these variants have the same framework but differ in the selection of the successor node (see below). The token circulates continuously through all the nodes of a mobile ad hoc network.

The main idea of this algorithm resides in the methods of choosing the next node to which the token will be sent. The variants are distinguished by the policies applied to determine the next node. They are divided into two classes, those using only local neighbourhood information (i.e. local algorithms) and those that uses information about all the network nodes (global algorithms). In order to protect against the potential loss of the token, the proposed algorithm uses TCP connection to deliver the token. For mobility, with some variants, nodes use “hello” messages to discover if neighbours remain connected to them. In each variant of the algorithm, the token carries with it some “count” information for each node in the system. The recipient node, before sending the token, uses the carried information to choose the next node to which it has to send the token (most of the variants chooses the next recipient, among those allowed, with the smallest value) and to update its own count information piggybacked by the token. The different methods are:

1. **Local-Frequency(LF) variant:** With this strategy, the token is sent to the least frequently visited neighbour of the token-holder node. To do so, the algorithm keeps track of how many times each node has been visited and this information is transported along with the token. The LF algorithm ensures that each node is visited at least once in the static connected topologies. However, the round length may increase without bound in certain topologies.

2. **Local-Recency(LR) variant:** In this case, the least recently visited neighbour is chosen. To implement this variant, for each node, a counter is associated which contain the time when the node was last visited by the token. As LF variant, there is no starvation in the case of static connected topologies, but, a round length of at most 2n (where n is the number of nodes) can be achieved.

3. **Global variants(GR and GF):** In these variants, the token is sent to the node that has been visited the least recently (in GR) or least frequently (in GF) among all the nodes of the network. The round length is equal to n for both the GR and GF variants. In GR variant, the nodes are visited in the same order but it is not necessarily true for the GF variant.

4. **Global variants with Next(GRN and GFN):** In these variants, all intermediate nodes in the route chosen for the destination are visited by the token. To do so, the token is sent to the neighbour of the token holder on the route to the one with the smallest count. The network layer
is then invoked to determine the neighbour on the route to a given destination. Simulation results only for GRN.

The performance evaluation is done with ns-2 simulator [38]. Each variant runs as an application on top of TCP, the Dynamic Source Routing Protocol [16], and IEEE 802.11 MAC [15]. The performance measurements considered are: The round length (recall that each node may be visited at least once in a round), message overhead which measures the number of bytes sent per round, (the overhead of sending control packets in medium access control is also included), and finally, time overhead which consists of time required to complete a round. The evaluation measures the average values of the metrics defined above. Other parameters concerning the network (number of nodes, the area in which nodes move, speed of nodes movements etc…) have been considered.

In static topologies, the best results in terms of number of nodes visited per round are performed by the GF and GR; This is of course by definition since the count concerns only the visited nodes and not the nodes that relay the token. In contrast, it is costly in term of bytes and time to reach the perfect round length. Good performance in terms of round length is exhibited by the LR algorithm which within one round converges to close to the optimal round length. Also, the simulation results show that for each variant of the algorithm, time and number of bytes per round have the same trends. The main difference is that the GF and GR variants are no longer optimal.

Among the global variants of the algorithm, the GF variant performs the best. The GRN’s performance is also comparable to the local variants and the GF variant.

In dynamic topologies, the LR variant continues to perform well results in all situations considered. However, the behaviour of the other variant of the algorithm becomes somewhat unpredictable in some cases.

4.2.6. Y. Chen and J. L. Welch Algorithm [10]

In[10], Y. Chen and J. L. Welch proposed a self-stabilizing mutual exclusion algorithm for mobile ad hoc networks. It is based on the algorithm LRV presented in [22], (which presents a good results in simulation,) and on the self-stabilizing concept defined par Dijkstra [12] and on the idea of “counter flushing” of [39]. The algorithm uses dynamic virtual rings (to reflect the changing topology,) formed by circulating tokens. It requires that the topology to be static while the algorithm is converging. But after it has converged, under a restricted mobility assumption, it guarantees both the safety and liveness properties of mutual exclusion.

5. Discussion

Due to the differences on the approaches adopted by the proposed algorithms, and the lack of methodology in complexity analysis for some of them, it is difficult to address a simple comparison. So, we limit our discussion to give some remarks about the presented solutions.

In [40], the proposed algorithm considers that node mobility is slow which is not realistic. In an other hand, it does not consider token loss and network partitioning and merging. For
performance analysis, a comparative study is made only by considering the same algorithm but on the top of TORA routing algorithm. So, more simulations are needed. In[41], the proposed algorithm acts with the same assumptions as [40] and also uses fewer messages which lead to a comprehensive execution. The simulation shows that the performance of the algorithm are comparable with those of Raymond’s algorithm. The other two algorithms[4][22] are not aware of the node mobility because of the use of the routing layer and consider that the network is not subject to partitions. The last algorithm[10] assumes a strongly connected network but links may change.

Moreover, Some of the solutions presented above consider that the links are uniform (bi-directional), but in reality the machines may be from different manufactures and consequently they may have different communication ranges, so the communication channels may be unidirectional. So, the solutions in [40][41] do not work in this case. One issue is to adapt these solutions to this new assumption.

6. Conclusion

In this paper, through the proposed solutions we have shown the progress of distributed mutual exclusion algorithms from static computing environment to mobile computing environment until mobile ad hoc computing environment. We mainly focussed on distributed mutual exclusion in mobile ad hoc networks. Among the presented algorithms, few ones [40][41] consider the node movement but with restriction. Others [4][22] function directly on top of a routing protocol, so they are not aware of the node movement (which is always restricted) because it is treated by the physical layer. It seems that the token based approach is suitable for mutual exclusion in mobile ad hoc networks. Finally, we consider that these algorithms represents a first step in giving solutions to a challenging problem: distributed mutual exclusion in mobile ad hoc networks.

References


