A distributed token based $h$-out of $k$ mutual exclusion protocol for mobile ad hoc networks

Mahfoud BENCHAIKA & Mohamed AHMED NACER

LSI-TR0208 Février 2008
A distributed token based $h$-out-of-$k$ mutual exclusion protocol for mobile ad hoc networks

Mahfoud BENCHAIBA$^1$ Mohamed AHMED NACER$^1$

$^1$LSI- Département Informatique, Faculté d’ Electronique & Informatique, USTHB
El Alia BP n°32, Bab Ezzouar, Alger, Algérie.
benchaiba@lsi-usthb.dz, anacer@mail.cerist.dz

Abstract: In this paper, we describe a new token based $h$-out-of-$k$ mutual exclusion protocol for mobile ad hoc networks. This protocol does neither use the routing layer nor a logical structure and agrees requests based on their distances away to the token, their olds, and there resources number. A request is sent on the routes of the nodes for which a request is present in the local queue, with a dynamical radius (typically 25%) computed, so to reach a part of them. Simulation results show that the radius tends to the value 1.2 for medium and high request loads. Freed resources follow in footsteps of the token. The protocol manages one token with $k$ resources in each partition and regenerates them in case of lost. Duplication of token and resources are avoided while partition’s merger. Simulation provides satisfactorily results in its entirety; the number of messages is rather low especially for medium and heavy loads.

Keywords: mobile ad hoc network, $h$-out-of-$k$ mutual exclusion, critical section, token, resource.

1. Introduction

Resources’ allocation is a fundamental problem in distributed systems. The mutual exclusion (ME) problem [3] is the entry point to this broad field. In this case, it is to manage a resource that can only be used by one process at a time; the code that uses this resource is named the critical section (CS). When the resource exists in $k$ copies, it is the problem of the $k$- mutual exclusion in which each process can use only one resource at a time. Therefore, at most $k$ processes perform their CS at the same time. However, the process may want to ask $h$ ($h \leq k$) copies of resources; so, the problem becomes the $h$-out-of-$k$ mutual exclusion ($h$-k-ME).

In order to solve the problem of $h$-k-ME, several solutions have been proposed in the literature of distributed systems. The first solution is due to Raynal [18]; it requires for a node to gather $k$ permissions to use the $h$ requesting resources (or simply to access its CS). All others solutions use the concept of quorum structures, which are collections of sets of nodes. [1] uses the concept of sets of arbiters, [2] use the k-arbiters; [13] uses the concept of local coteries; [9] uses the concept of k-coterie, [16] use h-k arbiters and [10] uses cohorts. Because the quorums construction are difficult, composition of operation on k-arbiters have been introduced in [12] and [11]. All these solutions have the same framework [15]: to achieve CS, a node must receive permissions from all members of the selected quorum; the difference between these various solutions is how to build qaurums. Such solutions generate a minimum cost of communication and some are fault-tolerant.
Quorums are not useful in the case of mobile ad hoc networks with high mobility since the communication topology is very dynamic; as a result, it is difficult to maintain a stable structure of quorums because it is not easy to contact all its members. The only known work in mobile ad hoc networks is due to J.R. Jiang [8]; his protocol is token based and is a result of adapting the solution to mutual exclusion for mobile ad hoc networks developed in [21] allowing it to inherit its main mechanisms. The solution of [21] is adapted from many others protocols: the routing protocol of [7] based on a tree, the protocol presented in [5], and from other ideas from [6] and [17]. This protocol uses a logical structure of a DAG mapped on the real topology of the network, with many paths leading to the token holder node. This solution is well suited for mobile ad hoc networks as it requires for nodes to keep information about only their neighbouring nodes. The DAG of token-oriented pointers maintains multiple paths leading to the node holding the token (i.e. the root). Requests are forwarded to the token holder along the tree. The token is delivered to the requesting node over the reverse path. The token movement and link downs involve a reverse link procedure to maintain the DAG in order to have always as root the token holder. This future supposes the absence of the network partitioning and nodes failures.

The solution of J-R. Jiang [8] uses only one token which is always owned by the root of the DAG. All nodes must have at any time at least one path to the root. When it losses this path a partial reverse link procedure must be executed. The main difference with the solution of [21] is related to the management of several resources. The token carries information on the number of free resources; furthermore, an additional message for the conveyance of the number of released resources to the token holder node is required. The node that receives the token and is the target can access its CS if the carried number of resources is enough to meet its request; otherwise it must wait the receipt of a liberation message in the hope to recover the missing resources. Before moving to its CS, a node looks to forward the token, with the remaining resources (even zero) to its next (if any). During the CS, if a node receives a request and still holds the token, it forwards the token to this node, which increases the concurrency to CS accesses. At the CS exit, a release message carrying the number of released resources should be sent on the token way.

The protocol of J-R. Jiang [8] does not take into account some characteristics of mobile ad hoc networks, such as failing and partitioning. In this paper, we propose a new distributed \(h-k\)-ME protocol for mobile ad hoc networks based on a token approach. This protocol does not rely on the routing layer and does not use logical structure for communication. A request is sent on the routes of the nodes for which a request is present in the local queue, with a dynamical radius (typically 25%) computed, so to reach a part of them. Furthermore, during token circulation, requests are agreed based on age, the distance to the token and the number of requested resources. Freed resources follow in footsteps of the token. The protocol treats many characteristics of mobile ad hoc networks: nodes movements (links ups and downs) partitioning and merger of the network. Nodes’ failures, token and resources’ loss may also happen. The protocol favours scalability of the network and considers an undetermined number of nodes. In addition, it manages one token with \(k\) resources in each partition and regenerates them in case of loss; duplication of token and resources are avoided while partition’s merger.

This paper is organized as follows: after a brief overview on some earlier works in section 1, section 2 presents the basic ideas of the protocol and the used communication structures. A description of the protocol operation is given in section 3. Section 4 completes the description of the protocol operation through a description of its behaviour with respect to the partitioning problem. A simple operation example followed by a discussion on some hidden aspects of the protocol is given in section 5. Section 6 deals with various simulation results obtained as well as their interpretations. Finally, section 7 concludes the paper.

2. Preliminaries

2.1. Network model

We assume a network consisting of a variable number of mobile nodes; each node has a unique identifier. Nodes movement are free (i.e. random mobility). The nodes use a wireless support to communicate to each others; communication links are bidirectional, FIFO and reliable. A node may at any time fail or leave the network. Node movements can lead to create and/ or loss of links, partitioning and the merger. The link layer allows nodes to know their neighbours at any time. Furthermore, we assume that the CS life is limited. Later in the paper and to describe our protocol, we are at the node designated by the index \(i\).
2. 2. Basic ideas

The protocol described in this paper takes some principles of the protocol developed in [4] sometimes with a few adjustments. The reading of this paper does not assume prior knowledge of the protocol developed in [4]. Our protocol uses the token approach by combining circulating and searching methods. The token is uniquely identified in each partition and carries, among others, a queue of requests collected during its travels and the number of available resources. To ensure the service safety, the holder of the token has the sole right to distribute resources to the requesting process. The protocol is based on local knowledge, that of the immediate neighbourhood and takes into account only the physical links to deliver information. That allows it to promote scalability of the network and to consider an unspecified number of nodes. On the other hand, the protocol makes use of a number of mechanisms that we describe in the future. Later in the paper, we use by misuse of language access to the CS instead of access to resources.

2.2.1. Sending radius

Since possession of the token is required for admission to the CS, circulating a request is a means to acquire it. To minimize the impact of the spread, we use the concept of sending radius. This radius is computed dynamically from the requests already received by i (in a local queue noted Q). Assuming that the received requests received are sorted locally in a temporary array T in increasing order of their roads length, the radius is computed as follows: \( R_i = \frac{1}{(T, \text{pos}) \cdot \text{Road}} \) where \( \text{pos} = \text{Round} (P^* / T, x) \), where P is a constant representing the proportion of nodes to reach and / x refers to the length of x. The nodes in the table T delimited by that radius will receive the request in the same way as those whose are in their paths. In figure 1(a), assume that the node 1 wants to make a request; it must compute its sending radius \( R_1 \). To do so, assume that its queue Q1 contains request of nodes 2, 4, 11 and 12 with respective routes 2; 5, 4; 6, 9, 11; 6, 10, 12. The request will be sent on these ways in order to reach all these nodes (i.e. 2, 4, 11 and 12) if \( R_1 = 3 \), or 50% of them if \( R_1 = 2 \). This method of radius computing fosters a cascade of roads to the token holder. Since the token gathers requests encountered on its way, all these nodes will eventually be satisfied, depending on mobility and node failures. In figure 1(b), the request of node 4 with route 2, 4 is registered at the node 3, the request of node 3 with route 9, 3 is recorded at node 8 and finally the request of node 8 is registered at node 7, i.e. the token holder node. So, node 7 is aware of the request of node 8. So, these all nodes can be meet in the order 8, 3 and 4. In general, the order of nodes’ satisfactions is not necessarily the same as that cited above; for this reason, we must carry the requests encountered by the token at visited nodes. Since the token intercepts requests encountered on its way, all these nodes will eventually be satisfied, depending on the mobility and failures of nodes.

2.2.2. Reviving of requests

A node requesting resources may not receive the token; this starvation has several reasons (due to mobility): frequent change in the neighbourhood of that node, frequent links changes in the network due to nodes failures, token loss, etc. In order to avoid starvation, the node requesting resources for which a request is not agreed after a limited timer (i.e. RequestTimer) has to revive its request. This revival is seen as another attempt of the same exclusion protocol for mobile ad hoc networks.

Legend
- : Ordinary node
- : Requesting node
- : Physical link

Fig. 1. Various examples.

2.2.3. Selection of the privileged node

Each node receiving a request will know the number of resources requested by the originating node of this request as well as the road travelled; then, it is inserted in its queue. If this queue is organized according to a policy that emphasizes closest requests, cases of starvation could result. Indeed, in the case of a group of nodes close to each other that require entry into CS frequently, the token will not cease to move between them, thus
depriving the farthest requests. To avoid this problem, the age of requests is also considered [14]. Moreover, in order to avoid the no efficient resources use, the number of resources is another criterion. Hence, the priority of the request of node $j$, enqueued at the privileged node $i$, is given by $Pr_j = (NS_j + nb_{hops} + Nb\text{-}req\text{-}resources_j$, $jj)$, where $NS_j$ is the stamp of the $j$’s request, $nb_{hops}$ is the number of hops between $i$ and $j$, $Nb\text{-}req\text{-}resources_j$ is the number of resources required by $j$ and the identity $j$ allows to create a total order between requests. As the distance between two nodes and the number of resources are limited and stamps of serviced nodes increase rapidly, priority necessarily turns for the more distant nodes. Note that when a node is selected as the next privileged node, the token is routed to it even if the number of resources is not enough to satisfy its request.

2.2.4. Resources’ locking

When a node is selected as the next to be served, the token is carried to it on the path of its recorded request. The intermediate nodes requesting for resources and lying on the road of the token toward the target can be served, since the token has reached them. This option avoids token to have the possibility to make several hops before reaching its destination, and consequently improves the requesting node waiting time. In order to avoid this option to influence the initial trajectory of the token, the notion of resources’ locking is introduced. When a node is selected as the next recipient, the number of resources requested by that node is deducted from the total number of free resources carried by the token. The intermediate nodes can only use the surplus, if any.

2.2.5. Token Road searching

During the token movement to a given destination, the road can be lost. Hence, a state is associated to the token; $ON\text{Route}$, if it normally follows a predefined route; $NOR\text{Route}$, if the road is lost and a new target node is not yet found. If a node $i$ receiving the token is requesting and is the target of the token or the token is in the state $NOR\text{Route}$, it vehicles the token to its next and enters its CS if the number of available resources can satisfy its request; if the number of available resources is not enough, it keeps the token locally with a local special resources’ waiting state ($Wait$). Otherwise, its role is reduced to route the token following the road carried with this token. If this road is broken, the node $i$ attempts to recalculate it by seeking in its neighbourhood the presence of nodes after the broken hop. If this operation fails, a new target for the token is selected according to the privileged node selecting method. If that fails, the token (with the state $NOR\text{Route}$) is sent to one of the neighbours in order to detect another target; the result is perhaps not the satisfaction of certain requesting nodes. This problem can be overcome by the possibility of requesting again the CS after a threshold.

The figure 2 shows the token route search. In figure 2(a), assume that the next targeted privileged node selected by the node 1 is 5. The token is sent to node 2 and is intended to follow the route 1, 2, 3, 4, 5 toward its destination with the state $ON\text{Route}$. Arriving to node 2 (figure 2(b)), links changes have happen; consequently, the next node (i.e. node 3) on the remaining token route is lost. The token route is recovered easily: As node 4 following the node 3 (i.e. the broken hop) on the token route is the neighbouring of node 2, then, the token re finds its route toward the node 5 and keeps the state $ON\text{Route}$. By contrary, in the figure 2(c) (where links changes have been occurred starting from figure 2(a)), the token arrives easily to node 3; following, a definitely loss of the token route to its destination has occurred. Another target must be found. Assume that the request of node 6 has been received by node 3; it is then selected by this node as next target and the token is sent to it with the state $ON\text{Route}$. The same problem as that of figure 2(c) has occurred in figure 2(d); In this case, the token rolls back in order to find a new target but with a state $NOR\text{Route}$.

2.2.6. Token research and regeneration

Token loss suspicion is triggered when a node does not receive it after a limited number of request’s attempts (i.e. $Request\_Threshold$). Beyond, if that node is not in the resources’ searching state (see section 2.2.9) (Otherwise means that the token is present and therefore the suspicion is aborted), a search throughout the partition is required. Because optimization reasons, the disseminated searching message has four major
objectives: token requesting, token searching, proposal to create a new token in the event of loss and request for
the resources’ annulment (see section 4.1). Thus all nodes, which are informed about this ongoing research,
block temporarily (do not request the CS and do not vehicle token and resources, if any). Aware that several
nodes can be simultaneously in the token searching situation, in the event of loss of this token (or its no-presence
in the current partition), a conflict occurs while its creation. To avoid this problem, we use a deterministic
mechanism of extreme selecting known in election protocols to choose the node which will create the token: for
example, “the node with the smallest identity”. Note that only nodes that seek the token participate actively
(may candidate) in the election process.

2.2.7. Token identification

Because of partitioning problems, several tokens may be created, one for each created partition. At partitions
merger, it is required to keep a single token in the produced partition. To facilitate the token remove, this latter
must be identified. The identity chosen for a token is that of the node which created it. But a node, having
created a token in a given partition may create another if it finds itself in another partition. In order to avoid the
ambiguity of token’s identification, each new token has also a sequence number local to the process which
created it.

2.2.8. Recovery of freed resources

Each node freeing resources is required to convey them to the token holder as free resources accompany the
token; so, it is good to follow them in the footsteps of the token in order to locate its owner by avoiding traffic
loop and optimizing the path when it is possible. To this end, each node keeps track of the node to which it has
sent the token last time. Because of links changing, this information alone is not enough to direct resources to
their destinations. In the example shown in figure 3, if the path of the token is 4, 3, 6, 5, 2, 7, and if a link down
occurs between nodes 5 and 2, resources issued, for example, from node 4 will follow the path 4, 3, 6, 5, 3; a
loop is then formed! Consequently, we introduce the concept of order number; the token tags its way through a
“circulating sequencer” assigned to it so that each node visited by the token keeps the last sequencer value
provided locally. When a node i releases resources, it links to the corresponding release message the local value
of the sequencer as an order number (or key); this key allows to identify uniquely the exploration done by this
message, given the multiplicity of release messages that explore at the same time the smooth path to their
destinations (see section 3.4). This right path is characterized by the growth in order numbers of nodes through
which a resources’ release message passes. So, with these two pieces of information, freed resources follow in
the best cases a linear path to the token holder (as in the case where each next node on the token path is still
close to its predecessor on the same path); otherwise, a tree path. Sometimes, and because link downs, these two
tools do not enough to deliver resources to their destination; then, if all roads are explored without success, the
resources are kept locally (either by the originating node of the liberation or by another achieved by these
resources) with a demerit state (i.e. Unable). This message waits then the establishment of a new link, which will
forward resources to the proper destination, or the passage of another resources’ release message.

2.2.9. Resources’ research and regeneration

Loss of resources can be suspected by a token holder node requesting the CS (i.e. state Wait) when it does not
receive the missing resources (of course, we accept that a request of a process does not exceed k resources) after
a deadline (i.e. ResourcesTimer); the reasons are either the resources’ loss due to the failure of one or more
nodes or their travel to other partitions or the loss of the way back due to frequent changes in communication
links. To solve all these problems, it is interesting to inform all nodes of the partition of this fact and to
regenerate the missing resources. To avoid the consequences of a false loss, the informed nodes are invited to
destroy possible resources in their possessions, even if they are in CS. A period of stabilization (i.e. RecoverTimer)
is necessary to all the partition nodes; it allows on one hand, for reasons of service safety, to the
token holder to delay nodes in CS to end before generating the resources, and for all nodes to neutralize any
possible token searching during this stabilization period.

Fig. 3. Loop circulating of resources.
2.3 Types of communication between nodes

The protocol described in this paper uses different types of communications to carry out the service of h-k-ME. A request for CS entry is the message expressed by Request (...); this message carries among others the identity of the requesting node, the request stamp, the number of request attempts, the requested resources’ number, the expected sending radius, and a knowledge on the nodes visited by the message. The privilege of CS entry is the message Token (...); this message carries the token identification, the identification of its former holder, the road to follow, information on its state (i.e. follows the planned path or has lost its way), the number of resources that were locked for the destination as well as information on requests known to the token. The released resources are sent to the token holder node by the message Release (...). This message carries the number of released resources, the key of the message and the order number of the sending node. Nodes failures, partitioning and frequent links changes between nodes can cause loss or temporary isolation of the token of the considered partition. The suspicion of token loss in this partition is therefore when a node requests the CS several times but without receiving the token; the message IsThereToken (...) allows searching a token. This message carries information on the request that has led to this research, and also an identification proposal for the token to create in the event of loss. If the token is not being lost, the holder uses the message ThereIsToken (...) as confirmation of its existence. The suspicion of resources’ loss in a partition is therefore when a requesting and token holder node has waited a long time without receiving freed resources it misses, the message IsThereResources(...) allows searching this resources. Mergers of partitions are also another consequence of the free movement of nodes; problems of token duplication can occur given that each partition has its own token. As the merger occurs at establishing links between nodes, exchange of information between the nodes in question becomes vital. To do so, the message LinkInfo (...) is used. At the level of a node issuer, this message carries identification of the token of its partition, and perhaps arguments of a message IsThereToken (...) or of a message IsThereResources(...), all received or issued by that node. Finally, in the case of a merger detection, the message DeleteToken (...) is used to delete one of the two tokens and all resources of a targeted partition. In order to inform nodes of the focused partition on the identification of the single kept token, its identification accompanies this message.

3. Protocol operation description

Each node in the network is running a symmetric algorithm; its text consists of a set of primitives associated to its different events. At each node, primitives run in an atomic manner, but events that occur during the CS execution can be treated.

3.1 Request propagation

A node i wishing to access the CS, if it is token holder it accesses it immediately if the number of resources is sufficient to meet its request; otherwise, it passes through a special local state (i.e. Wait) waiting for the resources’ liberation. A waiting period (i.e. ResourcesTimer) is associated with this expectation, after which the suspicion of resources loss is involved (see section 2.2.9). If the node i is not token holder, it sends a request Request (...), stamped as Lamport clocks [14] on the roads of a fixed proportion of nodes (for which a request is enqueued locally) defined by a radius computed locally (see the example in section 2.2.1). All intermediate nodes receiving this request have to vehicle it according to the indicated radius. In case where the requesting source node would have an empty queue or all requests are enqueued without roads (see section 5.2), these intermediate nodes (starting by its neighbours) have to compute a radius following the same process. A limited waiting delay (i.e. RequestTimer) is associated with the issued request. After that delay, if the token fails to reach the requesting node, the request is revived. In order to keep its seniority, the request is disseminated with the same stamp. In the worst case, this process is carried out a number of times (i.e. Request_Threshold) prior to suspect the loss of the token (see section 5.1).

When node i has to send a request but it is in a blocked state (see section 5.1), the request sending is postponed at the time of its unblocking.

3.2 Conditions to CS access

A requesting node i enters its CS if one of the following cases applies:
- it holds the token in the idle state and the resources’ number of the token is sufficient to meet its request,
- it holds the token and waits for resources (i.e. state Wait), but it has just received a release message of enough resources,
- it receives the token and more, either it is addressed and resources are enough, or not locked resources are sufficient, or the token has no destination address (i.e. state NORoute) and the carried resources are sufficient. In this case, resources eventually blocked at i are added to those of the token before acting on them.

In these above cases (except where i is interim), if the number of free resources is not enough, the node i stays (or remains, according to the case) in the state of waiting resources (i.e. Wait).

Note that when access to the CS is decided, if the node i is not blocked because of an ongoing token research, before running its CS, the token accompanied with remaining resources is sent to the next requesting node, if any. The next node may be the one selected according to the method described in section 2.2.3 or the one addressed by the route indicated on the token if i is not addressed; if the i’s queue is empty, the token is kept locally in the idle state. If the node i is blocked, sending the token is postponed at the time of unblocking.

3.3 Token movements

A node i holding the token and is not in a blocking state (see section 4.1) has to vehicle the token with ONRoute state as following:

- Case a: on the path to a new destination chosen according to the selection policy by locking the necessary number of resources (if this number is not enough, all these resources will be locked) if the following occurs:
  - The node i is token holder (and is in CS or is an idle state) and just received a new request,
  - The node i has decided to access into its CS and its queue is not empty,
  - The node i was blocked (and was waiting to send the token to the next requesting node), but has just been unblocked by the expiry of the timer StabilisationTimer (see section 4.1).
  - The token has just been received with a state NORoute, but the node i is not requesting the CS and its local queue is not empty.

- Case b: on the path of a predetermined destination, but after a possible path recovery operation (if it is lost because the moves of nodes planned on it) in the following situation:
  - The token has just been received with a predetermined route and the node i is not the target.

In the case (b), if the route recovery planned is impossible, a new target is chosen if any; otherwise, the token will become in the state NORoute and moves to a neighbour which is selected, in order to seek a new target avoiding loops (see the example shown in section 5.2). This latter operation is also done in the case (a), if the queue is not empty and paths to all selected nodes are not recoverable.

Apart from the cases cited above, the token is kept locally. It should be noted that for all cases mentioned above, if the node i is blocked because of an ongoing token research, sending the token is postponed at the time of its unblocking (see section 4.1).

3.4 Resources’ delivery

The resources’ delivery to the token holder is achieved by following the footsteps of the token (see section 2.2.8). To this end, a node i in possession of free resources, and which is not the token holder (otherwise, the resources are added to those of the token which implies a possible entry into CS if that node is awaiting resources (see section 3.2.), is called to convey a message of freeing resources to a neighbouring node. This neighbour may be the node to which it has sent the token last time; but if the link between them is broken, another neighbour is selected according to a criterion that avoids the message traffic loop (see example below). For each node i, two cases of delivery exist:

- Either when it exits its CS; in this case, the message will have as key the node’s i order number on the token path. If resources are blocked locally, for the sake of optimization, the liberation message will naturally convey the accumulation of resources freed and those blocked but with a key that takes the maximum among the key of the blocked resources and that of the freed resources.
- Or when it receives a resources’ liberation message; in this case, the key to send with the message is that of the received message if resources are not blocked locally; is the maximum of that of the blocked resources and of that of the received message, otherwise.

In both cases, the node i sends the message to its neighbour on the token way (ordinary delivery), if any and if the order number of the previous node (or that of i if the resources’ order number is that of i) on the path of the
Figure 4 gives an example of resources' delivery. In figure 4(a), it is assumed that from the node 1, the token is carried on the path 2, 4, 7, 3, 6, 5; and if the order number of node 1 is 10, the nodes traversed by the token have respectively as orders numbers 11, 12, 13, 14, 15, 16. In figure 4(b) where links changes took place, say that node 1 has to convey resources released locally; the corresponding message will have as order number 10 and as possible paths: 4, 2, 7, 2, 6, 5 or 4, 2, 6, 5 or 4, 3, 6, 5 or 3, 6, 5. For the first path (the longest one), as the next node on the token path (the node 2) is lost, the node 1 selects the node 4 (random choice), and sends it the resources’ message (in order to seek the right token path); the node 4, also for the same reason, chooses the node 2, which finally chooses the node 7. The node 7 with no other neighbours returns the message to 2, which chooses another (not explored) neighbour 6; hence, the message finds its way to the node 5 (the token holder node). For the latter path (the shortest), the node 1 chooses the node 3 instead of 4, so the message returns to the path of the token. In figure 4(b), if we suppose now that the path of the token starting from 7 is 2, 6, 3, 4, 3, 6, 5, and if the order number of node 7 is 10, the nodes crossed by the token have respectively as order numbers 11, 12, 13, 14, 15, 16, 17. The resources’ message issued from the node 7 with an order number 10 will be sent to 2, which passes it to 6, which sends it to its destination, i.e. the node 5 (which is the last node to which the token is sent by 6), it is an optimal path!

To support the tree path (i.e. without loop), we use a trace distributed in order to remember nodes from which each release message (known by its order number) is received at the first time.

To this end, the token has to carry information on the met requests; this solution creates an increase in the size of the token. One solution might be to remove each request according to a predetermined criterion (for example, when it reaches a certain period of stay in the queue). This solution is viable because mechanisms such as requests revivals, satisfaction of intermediate nodes, etc, can overcome its possible side-effects.

3.5 Queues updates

Updates of the two maintained queues (i.e. the queue $Q_i$ local to each node $i$ and that carried by the token) are done when receiving the token or a new request or a message $IsThereToken(..)$ or when leaving a CS. Queues updates allow to purge requests already fulfilled or older and to be aware of new requests or those with better routes. To this end, the token has to carry information on the met requests; this solution creates an increase in the size of the token. One solution might be to remove each request according to a predetermined criterion (for example, when it reaches a certain period of stay in the queue). This solution is viable because mechanisms such as requests revivals, satisfaction of intermediate nodes, etc, can overcome its possible side-effects.

3.6 Resources’ regeneration

This operation is performed on two separate occasions:

- When a resources requesting and token holder node (i.e. with state $Wait$) does not receive, after a fixed deadline (i.e $ResourceTimer$), the missing resources from releases made by other nodes. At this moment, that node (which is the only token holder of the partition) is called to broadcast a message $IsThereResources(..)$ in all its partition; this message is aimed to ask the nodes, holding resources blocked locally (i.e. state $Enable$) and those in use of resources, to delete these resources. Then, after a fixed stabilization period (i.e. $RecoverTimer$), that privileged node regenerates the $k$ new resources of the partition; as a result, the resources’ distribution resumes its normal course.

- At the moment of token regeneration: the operation of resources’ regeneration is also necessary. Indeed, the token lost or move towards another partition can take with it some resources, just as some nodes. The token regeneration operation is accompanied then by that of the resources. Thus, a node which creates the token is brought to regenerate $k$ resources. In order to ensure that only $k$ resources exist in the partition in progress, two recommendations are necessary: firstly, each node holding the "old" resources must remove them even if it is in CS; secondly, the distribution of generated resources is done after the flow of a fixed time (i.e. $RecoverTimer$). This delay allows nodes in use of resources to end their operation (which assures safety).
4. Partitioning and merger

Each partition is known by an identity (that of the token which circulates in it) which enables the detection of partitions mergers and thus keeps a single token. The token stamp is also used in order to distinguish partitions whose tokens have been created by the same node. This information is known by all the nodes of the current partition (see section 4.1).

4.1 Partitioning, token and resources’ loss

When the number of attempts of a token requesting node \( i \) reaches its limit, if it is not in the resources’ searching state (in which case the token is present), it begins to suspect the no presence of the token (i.e. partitioning or token loss), and changes the way to request the CS. Indeed, no sending ray is computed, its request will be broadcasted throughout the network; moreover, this request is no longer conveyed by a message \( \text{Request}(..) \), but by a message \( \text{IsThereToken}(..) \). The node \( i \) then blocks (does not generate any more request) pending the receipt of a message back \( \text{ThereIsToken}(..) \) during a period of time (i.e. \( \text{SuspicionTimer} \)), after which the partitioning and resources’ regeneration treatment is applied. If the message \( \text{IsThereToken}(..) \) meets the token at a given node, this node is required to distribute throughout the current partition the message \( \text{ThereIsToken}(..) \) but after a stabilization period (i.e. \( \text{StabilisationTimer} \)) to avoid blocking some nodes by messages \( \text{IsThereToken}(..) \) not extinguished, after their release. If before the expiration of \( \text{SuspicionTimer} \), the message \( \text{ThereIsToken}(..) \) is received, the node \( i \) unblocks and continues the dissemination of this message. However, if no response \( \text{ThereIsToken}(..) \) is received before the expiration of this timer, since its request implies also a proposal to create a new token, then the node \( i \), if it is only the one which requests the establishment of the token, creates it, regenerates \( k \) resources and the protocol restarts for this partition. In case of simultaneous proposals for the token creation, only one node, for example that with smaller identifier, leads (this method is similar to arrangements for the election on the basis of extreme), and the token created by that optimum node will have the identity of this node and as stamp the one proposed by this node. Nodes that do not create the token will take note of the new token and cancel the resources possibly in their possessions.

Recall that during the election process (i.e. broadcasting messages \( \text{ThereIsToken}(..) \)), each node \( i \) participating in the election offers its identity and a local sequence number as identification for the new token. At the end of the election (at the expiration of all timers \( \text{SuspicionTimer} \) at each node of the partition), variables that contain the identity of the candidate node and the stamp that suggested for the token (during the election process) now contain finals and unique values which are the identity of the elected node and the stamp it suggests for the token.

When a node \( j \) receives a message \( \text{IsThereToken}(..) \), it treats it as a normal request and if \( j \) is in the researching resources’ state, the current token research is stopped because the resources’ research comes from the token holder node! Otherwise, if \( j \) is not token holder, it sees this message as an exploration for the election to the creation of a token; this leads it to participate in this election by updating the appropriate information, and possibly continues the spread of this message and blocks (by setting its own timer \( \text{SuspicionTimer} \)) if it was not. In case \( j \) would be token holder (i.e. no partitioning), it is intended to address only the first message received. This treatment leads it to wait during a period of time bounded by \( \text{StabilisationTimer} \) in which \( j \) will make sure that the various simultaneous broadcasts of this type of message will be arrested (in this waiting period, \( j \) did not pass the token to the next). Then, it broadcasts the message \( \text{ThereIsToken}(..) \) throughout the network.

4.2 The merger

Partitions merger is detected at links creating. While the creation of a link, the two nodes forming it exchanges identifications of tokens associated to them. If token duplication is detected, one will be valid and the other will be deleted.

Once a node \( i \) detects a new neighbour, it transmits it a message \( \text{LinkInfo}(..) \) which contains information on its state in the following way: if \( i \) is not blocked (i.e. not looking for the token ), only information (identity and stamp) on the token used by \( i \) and the resources eventually blocked locally are transmitted; if \( i \) is blocked, the message \( \text{LinkInfo}(..) \) also includes information from the message \( \text{IsThereToken}(..) \) crossing or generated by \( i \) or information of the message \( \text{IsThereResources}(..) \) if contrary \( i \) is in the state of resources’ searching; this mechanism can address the case of momentary disconnection: the corresponding node can be token holder or blocking resources’ holder! It can be seen that for a link established, two messages are exchanged (by cause of symmetry), and therefore in the interest of reducing the number of messages \( \text{LinkInfo}(..) \), the following rule is imposed: if \( i \) is blocked, is resources’ searching or owns blocked resources, then it routinely sends the message

\text{Request}(..)
LinkInfo (..), but on the contrary if it is in a normal state, a single node (chosen by a deterministic manner) takes over the transmission of this message.

When a node \( j \) receives a message LinkInfo (..) from \( i \), if it finds that the token of \( i \) is the same as that of \( j \) (i.e. the link established is then internal to the partition), then according to the received information, \( j \) makes a treatment equivalent to that executed upon the receipt of a message IsThereToken (..) or Isthereresources(..) or Release(..).

However, if the tokens are different, then there was a merger and one token should be deleted; the token to delete may be for example, the most recently created in the case of tokens created by the same node or alternatively the one with the largest identity. If the token to remove is that of the \( i \)’s partition, \( j \) sends automatically to \( i \) an ordinary message LinkInfo (..) if it does not do it yet; otherwise, \( j \) disseminates the message DeleteToken (..) in its partition after emptying its queue and cancelling its eventual resources.

When a node \( r \) receives a message DeleteToken (..), if the token of the local partition is less prioritized, then it updates its local information with that received, resets its context (empties its queue, cancels its eventual resources, unblocks if it was blocked, offs its eventual timers, etc.) and continues disseminating this message.

Moreover, if \( r \) is token holder, it removes it even if it is in CS. This removal does not provide any problem because at the CS exit, there will be no token to vehicle. In all these cases, if \( r \) is requesting resources or in the state Wait, it sets the timer RequestTimer in order to revive its request later.

5. Operation example and discussion

5.1 Operation example

Figure 5 shows a scenario for the protocol operation in a way that facilitates the presentation; we note that actions which have no effects will not be cited. Initially (figure 5 (a)), the node 4 holds the token in the idle state with 12 free resources; nodes 2, 7 and 1 ask respectively 2, 2 and 5 resources with respective rays \( R_2 = 1 \), \( R_3 = 3 \) and \( R_1 = 1 \). Lets the stamps of all these requests are the same. Note that only requests of nodes 2 and 7 reaches the privileged node 4; as the two requests arrive with the same stamp, it is the number of hops increased by the number of resources that will determine the next privileged node, it is the node 2. The token is conveyed to it together with the free resources (i.e. 12); arriving at node 2, it serves its request and as requests of nodes 7 and 1 are both reached at the node 2, the next privileged node is 7; node 2 sends it the token on the way 1, 3, 7 by locking the requested resources and then accesses its CS. Arriving at node 1, the token allows it to serve its request as the number of unlocked resources (i.e. 8) is enough to agree its request; then, the token is forwarded on the intended way. When the token arrives at node 7, this node agrees its request and accesses its CS and keeps the token in the idle state (since its queue is empty). Then, the node 1 frees resources previously acquired; these freed resources follow the footsteps of token on the path 3, 7.

In figure 5 (b), the node 7 exits its CS and adds resources to those of the token which it holds, and then the node 4 makes a request on 11 resources with, for example, a radius \( R_4 = 2 \); the token is sent to it by the node 7 on the road 3, 4. By receiving the token, it sees that the number of free resources of the token (i.e. 10) is not enough to agree its request; then, it keeps the token and enters the state Wait (of missing resources). Thereafter, the node 2 frees its 2 resources; through the path traced by the token and the order numbers of nodes on this way that remained monotonous growing, the resources’ liberation message follows the path 1, 3, 4 instead of the long road 1, 3, 7, 3, 4 (thanks to the next node of node 3 on the path of token which becomes 4). The arrival of resources at the node 4 allows it to access its CS by keeping the token with \( I \) free resource.

In figure 5 (c), where links changes have occurred, nodes 7 and 3 require, respectively 5 and 6 resources (with \( R_7 = R_3 = 2 \)); assuming that the request of node 7 comes first at node 4, this node sends it the token on the road 3, 7. Arriving at the node 3, it can not agree its request since the number of resources surplus of the received token is zero, then it is limited to forward the token on its planned path. When the token arrives at the node 7, this node keeps it locally and enters to the state Wait.

In figure 5 (d), where other links changes have occurred, the node 4 exits from its CS, but losses its next on the path of the token (which is node 3), then it chooses a neighbour (say 5) and sends it the resources; the node 5, in the absence of another neighbour, returns the resources to the node 4, which chooses the other not explored neighbour (i.e. the node 1); the node 1 chooses the node 7, the holder of the token is then found. These resources enable the node 7 to access its CS sending the token to the node 3 which enters also into its CS and keeps the token with 1 free resource. Meanwhile, the node 1 requests 1 resource (with \( R_1 = 2 \)), then the token is received by this latter on the road 7, 1 allowing it access to its CS.

In figure 5 (e), where a partition took place, the node 5 requests 8 resources (with \( R_5 = 2 \)); the token is sent to it but without free resources, it enters into the resources’ waiting state by triggering the timer ResourcesTimer; the
node 7 exits from its CS, its message of freed resources explores the partition, for example, with the path 3, 6, 3, 2, 3, 7; the resources are kept by the node 7 to the state Unable. After then, the nodes 6 and 2 request each 4 resources.

In figure 5 (f), the node 7 breaks its link with the node 3 and forms a new link with the node 5. By exchanging information through messages LinkInfo (.), the node 5 receives 5 resources blocked at the node 7; The total number of resources acquired (which is 5 = (0 + 5)) is not enough to agree the request of node 5, it remains in the same state but resets its timer. Now suppose that the timer of node 5 expires, it broadcasts the message IsThereResources (.), the node 1, which is in CS cancels its resources. At the CS exit, the node 1 is without resources, it has not to send the resources’ liberation message! After the deadline of RecoverTimer, the node 5 regenerates all resources and accesses its CS. Alongside these operations, nodes 6 and 2 request several times the token without receiving it. Then, they proceed to the search for the token spreading the message IsThereToken (.). After timers SuspictonTimer, only the node 2 is required to create the token and regenerates k resources by accessing its CS; others nodes take note of the new token. “At the same time”, and at the expiry of its own timer SuspictonTimer, the node 3 is requested to cancel its resources. Thereafter, node 6 regenerates its request after the deadline RequestTimer it triggered when taking into account the new token; the token will be sent to it, allowing it access to its CS too (figure 5 (g)).

In figure 5 (g), the merger of the two partitions took place by the link up between nodes 7 and 3; only the node 3 sends a message LinkInfo (.) to 7 (since both nodes are in the normal state). Suppose that the token least priority is that of the partition of the node 7, a message DeleteToken (.) is broadcasted by the node 7 in its partition; this message allows the node 5 to remove its token with its 4 free resources, and cancels all the 8 resources in its possession.

![Diagrams](image-url)

---

5.2. Discussion

- How to agree requests carried by the token?
Recall that for each request encountered, the token carries only the identity of the node involved and the stamp of the request; roads carriage dramatically increases the size of the token and requires updating this road on each pass through a node. This implies that in the queue of each node, there may be presence of requests without roads. In the worst case to reach these nodes, the token will circulate through their research with a state NORoute; so, a tree travel of the network will be made. In the general case, the protocol provides other mechanisms to serve them (as intermediate nodes, further attempts requests…).

- How to avoid unnecessary token loop travel?
Given the example in figure 6(a): the token holder is the node 1 and the target of the token is the node 5 with the path 2, 3, 4, 5. In figure 6(b), when the token arrives at the node 3, the link with the node 4 is broken. The node 3 can not route token to its destination. Suppose also that it does not find a new target (i.e. its queue does not contain another request), so it gives the token to the last holder which is the node 2; the node 2 receives the token with a state NORoute, then it chooses a new target from its queue Q2, i.e. the node 6. It therefore refers the token to node 3 (which is on the road toward the new target) with a state ONSRoute. The node 3 still not having valid route, sends the token to its last holder which is node 2; a loop is then formed and the token circulates only between nodes 2 and 3. In order to avoid this situation, as soon as the node 3 returns the token to 2, the latter considers 3 as a node to avoid. It therefore chooses a destination other than node 3.
- Why the message LinkInfo(..) can carry in some cases information of a message IsThereToken(..) or IsThereResources(..)? Starting from the principle that the messages of the type IsThereToken(..) or IsThereResources(..) must be propagated throughout the network, a node may be isolated temporarily and therefore may not receive this message; when it joins the network, this spread is provided by carrying in the message LinkInfo(..) information of these messages.

- Why the message LinkInfo(..) can carry in some cases, the information of a message Release(..)? Given that the released resources can be sent (routed) to the token holder for future reuse, and since some nodes may have locally blocked resources, the discovery of a new neighbour gives them an opportunity to deliver these resources to their destination.

- Why regenerating resources instead of waiting for the receipt of blocked resources? During frequent disconnections, routes to destinations can be broken as frequently, which makes the recovery of not lost resources delicate, costly in time and nearly impossible in some situations. On the other hand, the lack of resources can mean either their loss, their blocking somewhere in the current partition, or both, and all this without any oversight. The fact that we must have at any time k resources in a particular partition, a solution leads to regenerate k resources at the single node holding the token. Therefore, the remove of resources blocked at certain nodes, resources which will be freed after CS exits and those in transit becomes an obligation. For precautionary reasons, the resources’ regeneration is done after a waiting period (i.e. RecoverTimer) in order to enable nodes currently using resources to exit from their CS.

- Why trigger the timer RecoverTimer at each node? In order to prevent temporary isolation and maintain at most k resources in a given partition, a node that receives the message IsThereResources(..) goes to the state of resources searching for a period of time limited by the timer RecoverTimer. This waiting time is required to inform all nodes of the partition (including those temporarily isolated) of this research. An isolated node is then informed by the message LinkInfo(..) received on the reinstatement of the network, which will allow it to remove resources eventually blocked locally!

6. Simulation

The protocol performance evaluation was carried out using the tool NS2 (Network Simulator 2) [20] version 2.26 under the LINUX operating system MANDRAKE 8.2. The protocol codification is performed using the object language Otel.

6.1. Simulation environment and parameters

The environment in which various simulations have been carried out can be described through a number of parameters. The fixed parameters are: the movement area is 1000x500 meters; the number of nodes in the network is 25; the CS duration is 0.1 seconds; the mobile nodes communication range is 250 meters; the simulation time is 100 seconds. The dynamic parameters are: the mobility model used is the random model based on the relative movement of nodes. This definition gives a good estimate of how the nodes are moving relative to each other. The mobility’s formula is provided below (3), $A_x(t)$ is the average distance between a node $x$ and all others at time $t$ (1); $M_x$ is the average mobility of node $x$ during the simulation (2) ($\Delta t$ is the computation step), $n$ is the number of nodes. From this formula, we have: low mobility: $0 < Mob <= 3$; average mobility: $3 < Mob <= 8$; high mobility: $8 < Mob$. For the simulation, we used four values of mobility: 0, 1.5, 5 and 10. With the various movement scenarios performed during the simulation, it is clear that the average connectivity varies between 5 and 7 neighbours by node.
A distributed token based h-out-of-k mutual exclusion protocol for mobile ad hoc networks

\[ A_s(t) = (1/(n-1)) \sum_{i=1}^{n} \text{Dist}(n_i, n) \]  
\[ M_s = (1/(t - \Delta t)) \sum_{t=0}^{T-D_t} \left| A_t(t + \Delta t) - A_t(t) \right| \]  
\[ \text{Mob} = (1/n) \sum_{i=1}^{n} M_i \]  

The load: It is defined by the Poisson process with parameter \( \lambda \) whose values taken for simulation are: 0.07, 0.10, 0.125, 0.167, 0.25, 0.5, 1 which correspond respectively to intervals \((1/\lambda)\) 15s, 10s, 8s, 6s, 4s, 2s, 1s during which a node, after agreement, generates a new request for entry into CS.

6.2. Results and interpretations

Before starting the study of the protocol performance, a series of preliminary tests revealed a number of values to timers for the proper functioning of the protocol. The values are: \( \text{RequestTimer} = 12 \) s; \( \text{SuspicionTimer} = 20 \) s: Time to vehicle the message \( \text{IsThereToken}(...) \) to all nodes added to the time for message receipt \( \text{ThereIsToken}(...) \) for all these nodes. If the value of the timer <20, then there will be a regeneration of the token, which is not tolerable. \( \text{ResourcesTimer} = 6 \) s, is the maximum time during which a node goes on hold message \( \text{Release}(...) \), to achieve a short synchronization time and to avoid spreading the message \( \text{IsThereToken}(...) \) by other requesting nodes; \( \text{StabilisationTimer} \) (resp. \( \text{RecoverTimer} \)) = 6 s, is the time it takes for the message \( \text{IsThereToken}(...) \) (resp. \( \text{IsThereResources}(...) \)) to reach all the network’s nodes. If the value of this timer is less than 4 seconds, some network nodes do not receive the message \( \text{IsThereToken}(...) \) (resp. \( \text{IsThereResources}(...) \)).

All performed measurements are divided into two categories: preliminary measurement and effective measurements. The preliminary measurements are aimed to quantify some behaviour of the protocol in order to effectively interpret the results related to effective measurements. The effective measurements are used of course to assess the performance of the protocol. All measurements were performed in the absence of partitioning. It should also be noted that all measurements (with the exception of those in figure 8) use the same legend for mobility than that of the figure 8; so, we describe it once for the sake of simplification.

Preliminary measurements

- The proportion \( P \): It represents the proportion of nodes, for which a request is present in the queue of the node wishing to enter in CS, to be achieved by its request. This parameter is determined using measurements on the number of messages by CS \( \text{nb_MSG / CS} \) for different loads and mobilities, and this by varying each time the proportion \( P \) (10\%, 25\%, .. 100\%) . The choice of \( \text{nb_MSG / CS} \) is motivated by the fact that in general the effective measurements are interdependent. Figure 7 shows that \( \text{nb_MSG / CS} \) is sensitive to variation in the percentage \( P \). The values 25\% and 50\% of the proportion \( P \) generate a minimum number of messages for mobility zero (figure 7(a)); in figure 7(b), all the proportions (except 100\%) provide a low number of messages. The values 10\% and 25\% of the proportion \( P \) generate a minimum number of messages for the average mobility (figure 7(c)); the values 25\%, 50\% and 100\% of the proportion \( P \) generate a minimum number of messages for high mobility (figure 7 (d)). It appears that the value 25\% of the proportion \( P \) is the most suitable according to the graphs in figure 7, this value is then used for all other simulation measurements.
Fig. 7. Graphs of the proportion $P$.

- The broadcast's rate ($nb\_diffusions / request$): This metric is related to the case of requests broadcasts when the request queues are empty. It is computed as follows: $number\ of\ broadcasts / number\ of\ requests$.

- The sending radius ($Radius$): This metric reflects the average number of hops that must travel a request; its computation is described in the section 2.2.1. In figure 8, the requests average sending radius decreases with the increasing of load and converges to $1.2$ hops for medium and heavy loads. This can be explained by the fact that over the request’s loads, the greater the number of requests in the queue is important; this implies that queues contains requests of closes nodes in terms of the number of hops, which causes a decrease in the radius. Note that the results of figure 9 are consistent with those in figure 8.

- The token’s road loss rate ($nb\_NORouteHops / nb\_TokenHops$): This metric reflects the number of times the token has lost the road to its destination (i.e. the number of sending with the state $NORoute$). It is computed as: $number\ of\ hops\ made\ by\ the\ token\ with\ the\ state\ NORoute / total\ number\ of\ hops\ made\ by\ the\ token$. According to the figure 10, the token road loss rate in the overall, tends to decrease with the increase of the load for all mobilities. On the other hand, the graphs of all mobilities are close to each others. All these results are due to the fact that over the load increase (i.e. the majority of nodes become requesting), then the token travels short distances in order to reach the next destinations. However, the token’s road loss rate converges to $30\%$. This convergence can be explained on one hand, by the fact that connectivity is relatively constant and on the other hand, by links breaking that accompany the nodes’ movements.
- The token’s research rate \((\text{nb\_IsThereToken} / \text{nb\_requests})\): This metric reflects the number of token searches by request. It is computed as: \(\text{number of times the message \text{IsThereToken} (...) is generated} / \text{number of requests}\). According to figure 11, the token’s research rate is generally quite low (6% maximum) and converges to a very low value whatever mobility. The low rate is because all nodes are part of the same partition, so usually a node which is isolated temporarily eventually joins the network and conveys the token possibly blocked locally.

- The resources’ research rate \((\text{nb\_IsThereResources} / \text{nb\_requests})\): This metric reflects the number of resources’ research per request. It is computed as: \(\text{number of times the message \text{IsThereResources}(...) is generated} / \text{number of requests}\). According to the figure 12, the resources’ research rate is very low and zero for low mobilities. The low rate is because all nodes are part of the same partition, so generally temporarily isolated nodes eventually join the network and convey the resources possibly blocked locally.

**Effective measurements**

- The number of messages per CS \((\text{nb\_MSG} / \text{CS})\): This metric is the average number of messages required for the CS achievement. It is computed as: \((\text{number of messages of CS requests} + \text{number of request reviving messages} + \text{number of hops made by the token} + \text{number of LinkInfo (..) messages} + \text{number of hops made by the message Release(..))} / \text{number of entries in CS}\). As shown in figure 13, all the graphs have the same allure, i.e. \(\text{nb\_MSG} / \text{CS}\) is decreasing and hence “inversely proportional” to the load. We also note some stabilization at medium and heavy loads (average 11 messages per CS). Furthermore, we can note that for medium and heavy loads, the impact of mobility change on \(\text{nb\_MSG} / \text{CS}\) is insignificant, in contrast to what happens with low loads where plots have different slopes. The reduction in the number of messages is jointly linked to the requests sending radius. That is, over the charge increase, the sending radius decreases; hence, the presences very likely of requests for nodes close each other in terms of hops number. Consequently, the token performs a minimum number of hops to satisfy nodes’ requests. On the other hand, as \text{Release (..)} messages follow the path of the token, they trail a short way too. These results are further confirmed by all preliminary measurements.
- The synchronization’s delay (Syn_Delay): It reflects the average number of hops made by the token and necessarily messages Release (...) between two CS. It is computed as: \((\text{number of hops made by the token} + \text{number of hops made by the Release (...) message}) / \text{number of entries in the CS}\). According to the figure 14, the appearance of graphs tends towards a common synchronization’s delay decreasing and converges to an average value of 2.5 hops. A reconciliation of graphs is observed at medium and heavy loads. In return, the influence of mobility is observed at low loads. By its very nature, the protocol seeks to satisfy the closest nodes (in number of hops). Also, each requesting node lying on the token road will be potentially satisfied and therefore, for heavy loads, the token and consequently messages Release (..) move most of the time between requesting nodes (which are geographically close each others); this explains the short synchronization’s delay whatever mobility. Regarding the low loads (from 0.07 to 0.25), the token circulates between not requesting nodes, in order to reach those who want to go to the CS. That said, more mobility, the higher the risk of links down and thus breaking roads (see figure 10) is more important, and therefore in these circumstances, the token will spend much more time looking for CS requesting nodes prior to achieve them; similarly, the messages Release (...) following in the footsteps of the token will go long ways; all these facts explain the increase in the synchronization delay.

- The satisfaction’s rate (nb_CS / request): This metric reflects the average satisfaction’s rate of requests for admissions in CS. It is computed as follows: \(\text{Number of entries in CS} / \text{number of requests}\), requests reviving have not been considered. For this metric, in figure 15 we see a rapprochement between different graphs with a speed nearly horizontal and converging towards a satisfaction’s rate of 1. However, the allure of graphs corresponding to low mobility and zero (0 and 1.7) shows a slight difference compared to the other two graphs, for the load ranging from 0.07 to 0.16. Particularly in the case of zero mobility, the satisfaction rate was slightly lower compared with other mobility. Finally, the overall impact of mobility is almost insignificant for medium and heavy loads. It should be noted that according to the operation traces, the majority of requesting nodes are satisfied. Indeed, the values of less than 1 of the satisfaction’s rate are mainly due to late requests, i.e. requests of nodes coming to the end of the simulation execution; they must wait for resources’ liberation. This is confirmed by the results of measurements on the rate of the token research and resources’ research that are practically zero. In other words, the vast majority of requests are not resorting to the search for the token (or resources), so they will be satisfied.

- The number of redundant messages per CS (nb_MsgRedundant / CS): This metric reflects the number of redundant messages per CS. It is computed as: \(\text{number of Request messages received} - \text{number of Request messages sent}\).
messages treated) / number of entries in CS. The appearance of graphs in figure 16 shows a common decrease in the number of redundant messages (with a tendency towards a degree of stability (0.75 hops)) with the load increasing. The graphs rapprochement between them shows a certain independence on mobility. Knowing that the rate of redundancy is directly related to the broadcasting rate, the graphs of figure 16 are consistent with the results in figure 8. Indeed, when the load is low, nodes tend to broadcast their requests; the high number of redundant messages is induced on the one hand, by the lack of control information being transported (i.e. information that avoids visiting nodes already visited) and on other hand, by the requests sending radius (see figure 9), which is high for small loads. For medium and heavy loads, the rate of redundancy weakens as the broadcasting radius decreases for these loads. In conclusion, it appears that the redundancy rate is mainly dependent on the broadcasting radius.

- Request reviving rate (nb_RequestReviving / CS): This metric reflects the number of request reviving relatively to total number of CS admissions. It is computed as: number of request reviving / number of requests. Naturally, we do not consider reviving in number of request. According to the figure 16, we see an approximation of the allure of graphs corresponding to different mobilities. However, we observe a trend towards value 0.5 from the medium to high loads for all mobilities. This rate shows that the majority of requests for admissions in CS are satisfied after 0.5 reviving. This amounts essentially to the time taken by a token holder node in waiting messages Release (...) to meet its request. In fact, and according to the operation traces, some requesting nodes keep the token blocked at home, while the waiting time for other requesting nodes flows and consequently they revive their requests for CS entry. Results of figure 17 also confirm that almost all requests are met (figure 15) but may be after revives.

6.3. Results related to the partitioning

The main measurement done is the number of CS executed after merger by using the token which will be removed and the number of messages Release (...) circulating in the partition containing the token to remove after merger. After each merger of two or more partitions, a single token remains valid and others are removed in a timely fairly quickly; in addition, at most $k$ resources circulate in the network, all others will be removed as soon receiving messages DeleteToken (...). Among the possible influences, we can cite the number of nodes in the concerned partition, the propagation speed of the message DeleteToken (...) and the location of the token and the resources to remove when the merger. Table 1 shows the number of CS performed after merger (using the token which will be removed). This table shows that the number of CS executed after merger is zero in most cases and takes 1 as high value. In addition, the number of messages Release (...) circulating in the partition that contains the token to be deleted is quite low. This result shows that this number depends mainly on the location of the token and the resources to be removed and also the propagation speed of the message DeleteToken (...) as compared to the CS duration execution.

To remedy this problem of temporary token duplication, a simple solution can be adopted: Before the merger (admit two partitions), each partition contains one token. When a link is formed and resulted in the merger of these two partitions, we can do so as “the merger has not yet been achieved” until the destruction of the token and CS exit of the last node in the targeted partition. To do so, communication between the two partitions is prohibited, by avoiding messages convey through this link during a fixed time. Other tests showed that each partition not containing token, is led to create one by consensus. This token creation came after deadlines (directly proportional to the propagation threshold). Also, each partition that contain the token and who suspects a lack of resources, leads to the regeneration of the latter after a finite time (bounded by ResourcesTimer and RecoverTimer).

<table>
<thead>
<tr>
<th>Number of nodes in the partition which contains the token to remove</th>
<th>4</th>
<th>7</th>
<th>11</th>
<th>15</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of CS executed during the merger (the merger has occurred during the CS executions)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>number of CS executed after the merger (before the message DeleteToken(...) reaches the token holder)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Number of messages Release (...) received before the reception of DeleteToken (...)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

| Tableau 1. Measurements related to the partitioning and the merger. |
7. Conclusion

In this paper, we presented a solution to the $h$ out of $k$ mutual exclusion problem for mobile ad hoc networks. The presented protocol is based on the network physical links and does not use logical structure. Requests are routed according to a dynamic sending radius determined by a local knowledge in order to cover a proportion $P$ of requesting nodes. The token is managed by a method that combines circulating and searching. In addition, the agreement policy of requesting nodes favours simultaneously closest nodes, oldest requests and less resources’ requesters (which avoids the no efficient resources use). Addressing nodes mobility through various mechanisms is an integral part of the proposed solution. Also, the protocol supports node failures, partitioning, merger and therefore, any problems that result (token loss, resources’ loss, token and resources’ duplications).

Based on various tests, as well as all metrics done, we can see that our protocol provides satisfactorily results in its entirety and are consistent with those obtained in the mutual exclusion framework [4]; the complexities of different metrics are roughly similar, but with added values generated by the messages related to the management of resources. The number of messages is rather low especially for medium and heavy loads. Our protocol is well suited to mobility, since the latter does not much on its overall performance. The number of messages sent per CS is rather low especially for medium and heavy loads because the ray converges to the value $I$. The synchronization delay depends on mobility but stabilizes as the network load increases and converges to the value $2.5$. The satisfaction rate tends to a stabilization value near to $I$ for medium and heavy loads, operation traces show that the not agreed requests are those which arrive just before the end of the simulation; they must wait for resources’ liberation. Also, the number of redundant messages falls with the increase in the charge. The requests reviving rate as a whole is relatively low when the load increases because a request for entry in CS is satisfied after $0.5$ number of reviving.

Acknowledgments

I would like to thank Miss A. Bousba and L. Souici for their cooperation in the realization of this work.

References

A distributed token based h-out-of-k mutual exclusion protocol for mobile ad hoc networks