A Group Mutual Exclusion Algorithm for Mobile Ad Hoc Networks

Ousmane THIARE
Department of Computer Science
LICP EA2175
University of Cergy-Pontoise-France
E-mail: othiare@dept-info.u-cergy.fr

Mohamed NAIMI
Department of Computer Science
LICP EA2175
University of Cergy-Pontoise-France
E-mail: naimi@dept-info.u-cergy.fr

Mourad GUEROUI
PRISM Lab
45, Avenue des Etats-Unis
78000 Versailles-France
E-mail: mogue@prism.uvsq.fr

***Abstract*- A mobile ad hoc network can be defined as a network that is spontaneously deployed and is independent of any static network. The network consist of mobile nodes[1] with wireless interfaces and has an arbitrary dynamic topology. The networks suffers from frequent link formation and disruption due to the mobility of the nodes. In this paper we present a token based algorithm for Group Mutual Exclusion in ad hoc mobile networks. The proposed algorithm is adapted from the RL algorithm in [1]. The algorithm requires nodes to communicate with only their current neighbors. The algorithm ensures the mutual exclusion, the bounded delay, and the concurrent entering properties.**

I. INTRODUCTION

A mobile ad hoc network can be defined as a network that is spontaneously deployed and is independent of any static network. The network consist of mobile nodes with wireless interfaces and has an arbitrary dynamic topology. The mobile nodes can communicate with only nodes in their transmission range and each one of them acts as router in routing data through the network. The network is characterized by frequent link formation and disruption due to the mobility of the nodes and hence any assumption about the topology of the network does not necessary hold.

Wireless links failure occur when nodes move so that they are no longer within tranmission range of each other. Likewise, wireless link formation occurs when nodes move so that they are again within transmission range of each other. In [1], an algorithm is proposed to solve the mutual exclusion problem for mobile ad hoc networks. The mutual exclusion problem is concerned with how to control nodes to enter the critical section to access a shared resource in a mutually exclusive way. The group mutual exclusion (*GME*) is a generalization of the mutual exclusion problem. In the *GME* problem, multiple resouces are shared among nodes. Nodes requesting to access the same shared resource may do so concurrently. However, if nodes compete to access different resources, only one of them can proceed.

In addition to the paper [1], there are papers proposed to solve mutual exclusion related problems for ad hoc networks. The paper [2] is proposed for solving the k-mutual exclusion problem, [4], for the leader election problem. There are several papers proposed to solve the *GME* problem for different system models. The papers [7][3][11] are designed for distributed message passing models, the paper [10], for self-stabilizing models. In this paper, we adapt the solution of [1] to solve the *GME* problem for mobile ad hoc networks.

The next section discusses related work. Section III presents our algorithm. We prove the algorithm correcteness in section IV. Conclusion and future work are offered in section V.

II. RELATED WORK

In [1], a token-based mutual exclusion algorithm, named *RL* (Reverse Link), for ad hoc networks is proposed. The *RL* algorithm takes the following assumptions on the mobile nodes and network:

1. the nodes have unique node identifiers,
2. node failures do not occur,
3. communication links are bidirectional and *FIFO*,
4. a link-level protocol ensures that each node is aware of the set of nodes with which it can currently directly communicate by providing indications of link formations and failures,
5. incipient link failure are detectable, providing reliable communication on a per-hop basis,
6. partitions of the networks do not occur, and
7. message delays obey the triangle inequality (i.e., messages that travel 1 hop will be received before messages sent at the same time that travel more than 1 hop).

The *RL* algorithm also assume that there is a unique token initially and utilize the partial reversal technique in [9] to maintain a token oriented *DAG* (directed acyclic graph). In the *RL* algorithm, when a node wishes to access the shared resource, it sends a request message along one of the communication link. Each node maintains a queue containing the identifiers of neighborings nodes from which it has received request for the token. The *RL* algorithm totally orders nodes so that the lowest-ordered nodes is always the token holder. Each node dynamically chooses its lowest-ordered neighbor as its outgoing link to the token holder. When a node detects a failure of an outgoing link and it is not the last outgoing one, it reroutes the request. If it is the last outgoing

[1] The terms processes and nodes will be used interchangeably throughout the paper.

T. Sobh (ed.), Innovations and Advanced Techniques in Computer and Information Sciences and Engineering, 373–377.

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 message to achieve the necessary change in their outgoing and incoming links and to reroute eventually their requests. So, the partial rearrangement is called. The algorithm guarantees the safety and liveness property ([12] for the proof).

Now we present the scenario for the *GME* problem. Consider an ad hoc network consisting of *n* nodes and m shared resources. Nodes are assumed to cycle through a non-critical section (*NCS*), an waiting section (*Trying*), and a critical section (*CS*). A node i can access the shared resource only within the critical section. Every time when a node i wishes to access a shared resource S_i, node i moves from its *NCS* to the *Trying*, waiting for entering the *CS*. The *GME* problem [8] is concerned with how to design an algorithm satisfying the following property:

- **Mutual Exclusion**: If two distinct nodes, say *i* and *j*, are in the *CS* simultaneously, then $S_i = S_j$.
- **Bounded Delay**: If a node enter the *Trying* protocol, then it eventually enters the *CS*.
- **Concurrent Entering**: If there are some nodes requesting to access the same resource while no node is accessing a different resource, then all the requesting nodes can enter *CS* concurrently.

Note that this property is a trivial consequence of *Bounded Delay*, unless runs with nonterminating *CS* executions are admissible.

For now let us focus on executions where all request are for the same node. Joung's informal statement of *concurrent entering* was that (in such executions) nodes should be able not only to concurrently occupy the *CS* but to concurrently enter it without "unnecessary synchronisation". This means that (in such executions) nodes should not delay one another as they are *trying* to enter the *CS*. Concurrent occupancy ensures that a node i *trying* to enter the *CS* is not delayed by other nodes that have already entered the *CS*. It does not, however, prevent i from being delayed (for arbitrary long) by other nodes that are simultaneously *trying* to enter the *CS*.

III. Proposed Algorithm

A *DAG* is maintained on the physical wireless links of the network throughout algorithm execution as the result of a three-tuple, or triple, of integer representing the height of the node, as in [9]. Links are considered to be directed from nodes with higher height toward nodes with lower height, based on lexicographic ordering of the three tuple. A link between two nodes is *outgoing* at the higher height node and *incoming* at the lower height node. A total ordering on the height of nodes in the network is ensured because the last integer in the triple is the unique identifier of the node. For example, if the height at node 1 is (2,3,1) and the height at node 2 is (2,2,2), then the link between these nodes would be directed from node 1 to node 2. Initially at node 0, height is (0,0,0) and, for all $i \neq 0$, *i*'s height is initialized so that the directed links form a *DAG* in which every non token holder has a directed path to some token holder and every token holder. The lowest node is always the current token holder, making it a sink toward which all request are sent. In this section, we propose a distributed algorithm to solve the *GME* problem for ad hoc mobile network.

In this algorithm, we assume that all the nodes concurrently accessing the same resource terminate their tasks. The algorithm is assumed to execute in a system consisting of *n* nodes and *m* shared resources. Nodes are labeled as $0,1,...,n-1$, and resources are labeled as $0,1,...m-1$. We assume there is a unique token held by node 0 initially. The variable used in the algorithm for node i are listed below.

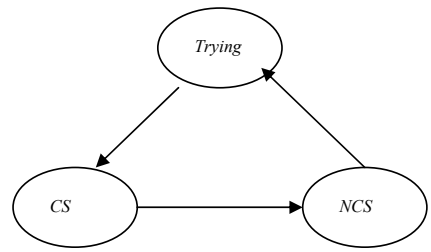

Fig. 1. States Process

- *status*: indicates whether node is the *Trying*, *CS*, or *NCS* section. Initially, *status=NCS*.
- *N*: the set of all nodes in direct wireless contact with node *i*. Initially, *N* contains all of node *i*'s neighbors.
- *Num*: counts the number of nodes within the critical section.
- *height*: a three-tuple (h_1, h_2, i) representing the height of node *i*. Links are considered to be directed from nodes with higher height toward nodes with lower height, based on lexicographic ordering. Initially at node 0, $height_0 = (0,0,0)$ and, for all $i \neq 0$, $height_i$ is initialized so that the directed links from a *DAG* where each node has a directed path to node 0.
- *Vect*: an array of tuples representing node *i*'s view of *height* of node *i*, $i \in N$. Initially, *Vect*[*i*]=*height* of node *i*. From *i*'s viewpoint, the link between *i* and *j* is *incoming* to node *i* if *Vect*[*j*]> $height_i$, and *outgoing* from node *i* if *Vect*[*j*]< $height_i$.
- *Leader*: a flag set to *true* if node holds the token and set to *false* otherwise. Initially, *Leader=true* if *i=0*, and *Leader=false* otherwise.
- *next*: indicates the location of the token from node *i*'s viewpoint. When node *i* holds the token, *next=i*, otherwise *next* is the node on an *outgoing* link. Initially, *next*=0 if *i*=0, and *next* is an *outgoing* neighbor otherwise.
- *Q*: a queue which contains request of neighbors. Operations on *Q* include *Enqueue*(), which enqueues