A Highly Adaptive Leader Election Algorithm for Mobile Ad Hoc Networks

Leila MELIT
Computer Science Department, University of Bejaia,
06000 Bejaia, Algeria
melitleila@yahoo.fr

Nadjib Badache
Laboratory of Computer Systems, USTHB,
Algiers, Algeria.
badache@mail.cerist.dz

Abstract – Leader election has been identified as a basic building block in distributed computing. MANETs are distinct from traditional distributed systems as they are dynamic and self-organizing networks because of their dynamic wireless link formation and removal, network partitioning and disconnections, limited bandwidth and energy and highly variable message delay. These characteristics makes the design of leader election protocols even more challenging than in classical distributed networks. In this paper, we present a leader election algorithm taking into account irregular topologies and mobility of the nodes. This algorithm elects as leader the node with the highest priority among the nodes in its connected component, where priority can be defined in a variety of ways. The paper also presents proofs of correctness to exhibit the fairness of this algorithm.

Keywords – leader election, mobile ad hoc networks, priority

1. INTRODUCTION

The leader election problem [1] is one of the fundamental problems in distributed computing. It has been widely studied since one reason for this wide interest is that many wired and wireless distributed protocols need an election protocol. For example, it is required in group communication service [2] key distribution and management [3] [4] and routing coordination [5].

In the context of mobile ad hoc networks, link changes are common and may cause the network to split into multiple connected components. Additionally, two connected components, each with its own leader, may merge. Moreover, in many situations, it may be desirable to elect a leader with some system-related characteristic as mentioned in [6]. Therefore, the elected leader should be the node which has the highest priority from among all nodes in its connected component, where the priority of a node is a performance–related characteristic such as the node's battery life, computational capabilities, etc. Thus, the requirements for a leader election algorithm becomes: "Given a network of mobile nodes each with a priority, after a finite number of topological changes, every connected component will eventually select a unique leader, which is the node of the highest priority from among the nodes in that component".

The aim of this paper is to propose a higher priority index–finding algorithm that can work in highly dynamic and asynchronous mobile ad hoc networks.

The rest of this paper is organized as follows: the next section describes our leader election algorithm for mobile ad hoc networks. The correctness proof of the algorithm is given in Section 3 and Section 4 concludes this paper.

2. LEADER ELECTION ALGORITHM

Figure1 shows our algorithm proposed to solve the election problem in mobile ad hoc networks. Each mobile node of our system has two possible states: leader or candidate. A node passes in candidate state if it receives a leader message from a node with higher priority. But it can return to leader state when it detects the departure of its leader.

```
1.  Leader IDi ← IDi;
2.  LeaderPriorityi ← Priorityi;
3.  Statei ← leader;
4.  IDElecMobilei ← 0;
5.  IDElecLeaderi ← IDElecMobilei;
6.  Start Timeri;
7.  Broadcast leader (LeaderIDi, LeaderPriorityi, IDElecLeaderi);

8.  Upon expiration of timeri do
9.      If (Statei = candidate) then
10.         Leader IDi ← IDi;
11.         LeaderPriorityi ← Priorityi;
12.         Statei ← leader;
13.     IDElecMobilei ← IDElecMobilei + 1;
14.     IDElecLeaderi ← IDElecMobilei;
15.     Broadcast leader (LeaderIDi, LeaderPriorityi, IDElecLeaderi);
16.     Restart Timeri;

17. Upon reception of message Leader (LeaderIDj, LeaderPriorityj, IDElecLeaderj) do
18.     If (LeaderPriorityi < LeaderPriorityj) or ((LeaderPriorityi = LeaderPriorityj) and (LeaderIDi > LeaderIDj)) then
19.         LeaderIDi ← LeaderIDj;
20.         LeaderPriorityi ← LeaderPriorityj;
21.         IDElecLeaderi ← IDElecLeaderj;
22.         If (Statei = leader) then
23.             Statei ←candidate;
24.         Broadcast leader (LeaderIDj, LeaderPriorityj, IDElecLeaderj);
25.         Restart Timeri;
26.     Else
27.         If ((LeaderPriorityi=LeaderPriorityj) and (LeaderIDi=LeaderIDj) and (IDElecLeaderi < IDElecLeaderj)) then
28.             IDElecLeaderi ← IDElecLeaderj;
29.             Broadcast leader (LeaderIDj, LeaderPriorityj, IDElecLeaderj);
30.             Restart Timeri;
```

Figure 1: A highly adaptive leader election algorithm for mobile ad hoc networks.

The competition takes place only between nodes that are in a leader state. Each node that lost becomes in the candidate state. Our algorithm ensures the leader of each connected component to be the unique node in its connected component in leader state. The other nodes in its connected component must be in candidate state.

Assumptions on system, mobile nodes and networks are:

- Each node has unique identifier that is fixed throughout the node's lifetime.
- Each node has a priority associated with it. The priority of a node indicates its "attractiveness" as a leader of the network, and can be any performance–related attribute.

Each node *pi* maintains the following data structures:

- *IDi*: indicates the identifier of the node.
- *Priorityi*: indicates the priority of the node.
- *LeaderIDi*: indicates the leader's identifier
- *LeaderPriorityi*: *integer*; indicates the priority of the leader.
- *Statei*: indicates the state of the mobile node i. It can take two possible values: Leader or candidate.
- *Timeri*: represents the maximum delay that pi expects until it receives a message from the leader of the connected component that it belongs.
- *IDElecmobilei: indicates the identifier of the latest leader message originated by i.*
- *IDElecLeaderi: indicates the identifier of the latest leader message originated by the leader of i.*

We say that the leader of a node i is lower priority than j's one if and only if: (LeaderPriorityi < LeaderPriorityj) or ((LeaderPriorityi = LeaderPriorityj) and (LeaderIDi > LeaderIDj)).

When a node triggers an election, it elects itself, informs its neighbors and starts its timer. The expiration of the timer for a node in the leader state means that it hasn't received a leader message from a node with higher priority. So it continues its participation in the competition by sending a leader message containing its information. However, the node in the candidate state, when its timer expires, it realizes that the leader is absent and triggers a new election.

Each node i upon receiving a message Leader (LeaderIDj, LeaderPriorityj, IDElecLeaderj) compares the priority of its leader with the priority of LeaderIDj indicated by LeaderPriorityj. If its leader priority is the lowest, it updates its information by those transported by the received leader message, turns its state to candidate only if it was in the Leader state, restart its timer and rebroadcast the same message received. If the leaders of the nodes i and j have the same priority, but IDElecLeaderi < IDElecLeaderj, then the node i puts its IDElecLeaderi to IDElecLeaderj, rebroadcasts the same message that had received and restarts its timer. The comparison between IDElecLeaderi and IDElecLeaderj allows the algorithm to drop old messages.

Our algorithm is can tolerate node mobility, network partitioning and merging components. For example, if the leader breaks down, then after a bounded time, each node detecting the leader departure elects itself as leader and broadcasts its identifier with its priority (lines 8-16). If this failure causes partitioning of the component in two other components, each component will have eventually a unique leader. Moreover, if a node other than the leader breaks down and causes the partitioning of the component, then nothing will change in the component that contains the leader, but the other component elects a new leader and its identifier is propagated throughout that component. The detection of partition from the leader is guaranteed by the use of the timer.

3. CORRECTESS

We assume that T is the moment when the topology becomes static. We also assume that Nmax is the node that has the highest priority value in its component. The following theorems establish the correctness of the leader election algorithm proposed above.

Lemma 1. There is a time after which Nmax permanently satisfies that leaderNmax = Nmax and broadcasts periodically a message leader (LeaderIDNmax, LeaderPriorityNmax, IDElecLeaderNmax).

Proof:

Lemma 1 means that after time t> T, node Nmax will not receive any message leader (LeaderIDj, LeaderPriorityj, IDElecLeaderj) such as (LeaderPriorityj > LeaderPriorityNmax) or ([LeaderPriorityNmax = LeaderPriorityj] and [LeaderIDj < Nmax]). Therefore, it will not execute the lines (17-25) of the algorithm, and the property leaderNmax = Nmax is always satisfied at every time t> T. To see that it is satisfied, we distinguish the following cases:

Case 1: There was no node i in the Nmax's component such as (LeaderPriorityi > LeaderPriorityNmax) or ((LeaderPriorityNmax = LeaderPriorityi) and (LeaderIDi < Nmax)). In this case, lines (17-25) of Task 1 had never been executed by the node Nmax. LeaderNmax = Nmax is always satisfied and will always be satisfied at every time t> T.

Case 2: There was at least one node i in the component such that (LeaderPriorityi > LeaderPriorityNmax) or ((LeaderPriorityNmax = LeaderPriorityi) and (LeaderIDi < Nmax)) which sent periodically the message leader (LeaderIDi, LeaderPriorityi, IDElecLeaderi) by executing line 15 of the algorithm. In this case, leaderIDNmax = i has been satisfied after execution of Line 19. Then, after a finite number of topology changes (after a time t> T), the node i (and definitely all nodes that are higher priority than Nmax) left the component or crashed. Therefore, Nmax becomes the node that has the highest priority in its component. We distinguish two sub cases.

a. After time t> T, the node Nmax will not receive any message leader (LeaderIDi, LeaderPriorityi, IDElecLeaderi) such as (LeaderPriorityi > LeaderPriorityNmax) or ((LeaderPriorityNmax = LeaderPriorityi) and (LeaderIDi < Nmax)). So, lines 19-25 will not be executed, the timer will not be restarted at line 25 and it will finally expired (line 8) and leaderIDNmax = Nmax will be satisfied (line 10).

b. After time t> T, node Nmax receives a message leader (LeaderIDi, LeaderPriorityi, IDElecLeaderi). It is an old message originated by node i and is still circulating in the component after the departure of i. But when the node Nmax receives this message for the second time, condition in line 27 will never be satisfied (because IDElecLeaderi is the same for the two messages) and so lines 28-30 will not be executed. Therefore, Nmax will not restart its timer, this later will expire. Consequently, lines

8-16 will be executed and leaderIDNmax becomes Nmax.

Finally, as leaderIDNmax = Nmax, Nmax broadcasts permanently and periodically the message leader (LeaderIDNmax, LeaderPriorityNmax, IDElecLeaderNmax).

Lemma 2. There is a time after which every message leader (p, LeaderPriorityp, IDElecLeaderp) with p ≠ Nmax disappears from the system.

Proof:

Note that initially, each node p begins the execution by electing itself as leader. i.e., leaderIDp = p (line 1). As long as it remains leader, it broadcasts leader (LeaderIDp, LeaderPriorityp, IDElecLeaderp) (line 15). Also note that each node p on receipt of each message leader (LeaderIDj, LeaderPriorityj, IDElecLeaderj), makes leaderp = j if j is higher priority than p (lines 17-25).

At a time t> T, Nmax is the node that has the highest priority value in its component and it periodically broadcasts the message leader (LeaderIDNmax, LeaderPriorityNmax, IDElecLeaderNmax) (Lemma 1). Each node p ≠ Nmax, upon receipt of this message makes leaderIDp = Nmax (line 19) and sends leader (LeaderIDNmax, LeaderPriorityNmax, IDElecLeaderNmax) to all its neighbours as shown on line 24 of the algorithm.

After time t> T, the timer of p will not expire as Nmax periodically broadcasts the message leader (LeaderIDNmax, LeaderPriorityNmax, IDElecLeaderNmax). So p will never execute line 15 of the algorithm (the message leader (LeaderIDp, LeaderPriorityp, IDElecLeaderp) will never be sent).

Finally, every message leader (LeaderIDp, LeaderPriorityp, IDElecLeaderp) with p ≠ Nmax was disappeared from the system.

Theorem. There is a time after which each node p in the same component have leaderNmax = Nmax, where Nmax is the node that has the highest priority value in that component.

Proof:

Lemma 1 shows that there is a moment after which the node Nmax maintains leaderNmax = Nmax (ie it is the leader of its component) and periodically broadcasts the message leader (LeaderIDNmax, LeaderPriorityNmax, IDElecLeaderNmax) to notify the other nodes in its component that it is the leader. Lemma 2 shows that no other message will circulate into the component. So the only message that is circulating is the message leader (LeaderIDNmax, LeaderPriorityNmax, IDElecLeaderNmax) and all nodes of the component have Nmax as a leader.

So, the algorithm ensures that:

For each component C in ad hoc network, there is a node Nmax in C and a moment after which, for every node p in C, leader p = Nmax which is the node of the highest priority in C.

4. CONCLUSION

In this paper, we have proposed a leader election algorithm for ad hoc networks that can tolerate intermittent failures, such as link failures, sudden crash or recovery of mobile nodes, network partitions, and merging of connected network components. This algorithm ensures that every connected component will eventually select a unique leader having the highest priority. To elect a unique leader, the algorithm requires mobile nodes to communicate only with their immediate neighbors. Our future plan is to simulate the algorithm to study self-stabilization propriety witch is very important in mobile computing.

5. REFERENCES

[1] N. Lynch, “Distributed algorithms”, Morgan Kaufmann Publishers, 1996.

[2] G.-C. Roman, Q. Huang, and A. Hazemi, “Consistent group membership in ad hoc networks”, Proceeding of the 23rd International Conference on Software Engineering (ICSE ’01), 2001, pp. 381-388.

[3] N. Asokan and P. Ginzboorg, “Key agreement in ad hoc networks”, Computer Comm., vol. 23, no. 17, 2000, pp. 1627-1637.

[4] B. Lehane, L. Dolye, and D. O’Mahony, “Ad hoc key management infrastructure”, Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC ’05), vol. 2, 2005, pp. 540-545.

[5] C. Perkins and E. Royer, “Ad-hoc On-Demand Distance Vector Routing”, Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, New Orleans, LA, February 1999, pp. 90-100.

[6] S. Vasudevan, J. Kurose, and D. Towsley, “Design and analysis of a leader election algorithm for mobile ad hoc networks”, Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP’04), Octobre 2004, pp. 350-360.