Top K-leader Election in Wireless Ad Hoc Networks

Vaskar Raychoudhury, Jiannong Cao, Weigang Wu

Department of Computing, The Hong Kong Polytechnic University, Kowloon, Hong Kong
{csvray, csjcao, cswgwu }@comp.polyu.edu.hk

Abstract

In this paper, we propose a distributed algorithm to elect the top K leaders among the nodes in a wireless ad hoc network. Leader election is a fundamental distributed coordination problem arising from many applications, e.g. token regeneration, directory service. However, there is no deterministic algorithm proposed for electing k leaders. In our algorithm, election is based on the weight values of the nodes, which can represent any performance related attribute such as the node's battery power, computational capabilities etc. To achieve message efficiency, coordinator nodes are first selected locally and then the coordinator nodes collect the weight information of other nodes using a diffusing computation approach. The coordinator nodes collaborate with each other to further reduce the message cost. Node failures are also considered in our design. The simulation results show that, compared with a naive solution, our proposed algorithm can elect top K leaders with much less message cost.

1. Introduction

A wireless ad hoc network is a collection of computing devices or nodes that can communicate via message passing over wireless links. Nodes that are in transmission range of each other can communicate directly otherwise, they communicate via WHITE nodes. Nodes may crash anytime giving rise to cessation of communication between two previously connected nodes.

Due to the characteristics of wireless ad hoc networks, including mobility, communication and resource constraints, leader election in ad hoc networks is a challenging problem.

The leader election problem arises from many distributed applications that require one or more nodes to act as coordinators, initiators, or perform some special role, e.g. directory server or token regenerator. Typically, the leader election problem is to elect a unique node to play a particular role [7].

In this paper, we study the weighted *K*-leader election problem in wireless ad hoc network environments, which aims to elect a number *K* of leaders with the top *K* highest weight values. The weight is an abstract property and can be used to refer to any wanted attribute of a node, e.g. remaining battery power, memory size, etc. More precisely, we define the problem using two correctness properties:

- *Liveness:* eventually *K* nodes are selected as the leaders.
- *Safety*: each of the elected leaders should be among the top *K* nodes according to the weight value.

Several solutions have been proposed to solve the leader election problem. However, existing leader election algorithms can not satisfy the requirements of election top *K* leaders in a wireless ad hoc environment.

Algorithms proposed in [1][2][12] are designed for wired networks while the algorithm in [9] is for infrastructure networks where the election is in fact done by the wired part of the network.

Several leader election algorithms [4][13][7][6][8] have been proposed for ad hoc networks but they also consider only the election of one unique leader.

To our knowledge, the only work on *K*-leader election is done by Ferreira et al. [4]. They designed two *K*-leader election algorithms to solve the problem of duplicate elimination in storage systems. However, their algorithms are probabilistic and the election is based on the ID of a node.

Compared with existing work, our proposed algorithm has the following two features. First, we do not assume a unique weight value for each node. In our design, nodes can have same or different weight values. Such an assumption is reasonable and necessary for electing nodes with some desirable property, e.g. memory space size.

Second, except one-hop neighbors, we do not assume a node know other nodes in the network. This is inspired by the observation that an ad hoc network is high dynamic and auto-configured and a node knows all other nodes in the network is infeasible.

With the above two features, much more challenges arise in the design of leader election algorithm and existing solutions can not work at all.

To evaluate the performance of our proposed algorithm, we have carried out extensive simulations and present our results with in-depth analysis. Our simulations show that the algorithm works efficiently even in the scenarios with a high node failure rate.

978-1-4244-2390-3/08/$25.00 ©2008 IEEE

The rest of the paper is organized as follows. Section 2 reviews existing leader election algorithms, including both the election of one leader and K leaders. Our proposed top K leader election algorithm is presented in Section 3, including data structures and operations. We also discuss how to handle node failures in the end of this section. The correctness proof of our algorithm is presented in Appendix I. Section 4 reports our performance evaluation results. The performance of our algorithm is also compared with similar ones. Finally, Section 5 concludes that paper and points out directions of future work.

2. Related work

Leader election algorithms for wired networks can be found in [1][2][3]. The algorithm in [9] is for infrastructure networks where the election is in fact done by the wired part of the network using Garcia Molina's *bully* algorithm [5].

Leader election algorithms for ad hoc networks are proposed in [11][7][6][8][13][4]. Based on the routing protocol named TORA [10], Malpani et al developed an election algorithm [7]. Hatzis et al [6] proposed two leader election algorithms based on the geographical positions of the MHs. Two randomized algorithms for single-hop and two-hop single channel MANETs are presented in [8]. In these algorithms, the leader is elected by the collision state of the channel in which the MHs emitting signals randomly and synchronously. Sundramoorthy et al. [11] design a leader election algorithm to solve the dynamic directory election problem in service discovery environment.

The algorithm proposed in [13] adopts the well-known diffusing computation approach proposed by Dijkstra and Scholten [3]. Although, we also use the diffusing computation approach in our design, our algorithm aims to elect K leaders in stead of a single leader. Although the authors of [13] claim that their algorithm can be extended to elect top K nodes, their approach is not efficient in terms of message cost, especially in large ad hoc networks. Moreover, following their scheme, every WHITE node has to sort the node values to select the top K child node which is prohibitive for some low weight node.

Two K leader election algorithms are proposed by Ferreira et al. [4]. They use the K-leader election algorithm to solve the problem of duplicate elimination in storage systems. However, the two algorithms proposed in [4] are probabilistic solutions and they are not designed for wireless networks.

3. The proposed election algorithm

Our proposed algorithm adopts the approach of diffusing computations [3] to perform leader election. Informally, the algorithm operates as follows. First, a node with the highest weight among its 1-hop neighbors is voted as a RED node. The nodes are not voted as RED nodes are called WHITE nodes. Then, the RED nodes start the diffusing computation procedure concurrently. To save message cost, a WHITE node is allowed to be included in the diffusing computation of only one RED node. When the diffusing computation terminate, the results collected by different RED nodes are merged and eventually the highest weight RED node will get the complete weight information of all the nodes in the network.

3.1 System model and assumptions

The leader election problem is considered in a wireless ad hoc network that consists of a set of n (n>1) nodes, each of which has a unique ID. The nodes communicate by sending and receiving messages through wireless channels. A node only knows the IDs of its neighboring nodes. Whether two nodes are neighbors, i.e. they are directly connected, is determined by the signal coverage range and the distance between the nodes. Each node is a router and the communication between two nodes can be multiple hops. A node may fail by crashing, i.e. prematurely halting.

Each node has a weight associated with it. The weight of a node indicates its ability as a leader of the network and can be any performance related attribute such as the node's battery power, computational capabilities etc. Two nodes may have the same weight value.

3.2 Data structures and message types

When executing our algorithm, each node i maintains necessary information about its state, which is stored in the following data structures.

wt_i: Weight of node i

$color_i$: Color of node i = {*WHITE, RED*}

$visitor_i$: Identifier of RED node which visits node i

nbr_i: Set of immediate (1-hop) neighbors of node i

$pred_i$: Predecessor of node i (from whom i receives the first SEARCH message)

$ResultQ_i$: List of nodes visited by a node i

$RedQ_i$: List of RED nodes known to node i

LDR: List of K highest weight nodes along with their corresponding weights.

The following types of messages are exchanged between nodes in our algorithm.

ELECT(i,wt$_i$): the message sent by node i to its neighbors

VOTE(i): the message sent by a node to vote for its neighbor i.

SEARCH(i): the message initiated by a RED node i and propagated among other nodes to collect the weight values.

ACK(): the message sent by a node *j* to inform another node *k* that they are both visited by the RED node *i*.
NACK(i, wt_i): the message sent by a node *j* to inform another node *k* that *j* is visited by RED node *i*, different from the one that visited *k*.
SIGNAL($RedQ_j$, $ResultQ_j$): the message sent from node *j* to its predecessor in response of the SEARCH message received.
RESULT(P, $ResultQ_j$): the message sent by node *j* to the highest weight RED node *i* carrying $ResultQ_j$. *P* is the set of RED nodes redirected to *i*.
DIRECT(i, wt_i): the message sent by node *j* to all the RED nodes known to *j* except the highest weight one.
LEADER(LDR): the message sent by the highest weight node to all the other nodes in *LDR* to inform the leadership of the elected nodes.

3.3 Operations of the proposed algorithm

We divide the whole algorithm into three phases. Phase I is used to select RED nodes and Phase II is the diffusing computation procedure for RED nodes to collect weight values. Finally, Phase III merges the weight values collected by different RED nodes and elect the top *K* leaders. The pseudocode of the three phases is shown in Fig. 1, Fig. 2 and Fig. 3.

```
//The code executed by each node, i
(0) color_i ← WHITE, pred_i ←NULL, visitor_i ←NULL,
                out_i ← 0, RedQ_i ←NULL, ResultQ_i ← NULL;
(1) send ELECT (i, wt_i) to all the neighbors in nbr_i;
(2) wait until an ELECT message is received from each neighbor;
(3) send VOTE(i) to j, where j is the the highest weight node in nbr_i;
(4) wait until a VOTE received from each node in nbr_i or
        a SEARCH message is received;
(5) if (a VOTE received from each node in nbr_i) color_i ←RED;
```

Fig. 1. Pseudocode of Phase I

3.3.1 Phase I

Initially every node in the network sets its color to WHITE. Every node then sends an ELECT message containing its weight to all its 1-hop neighbors. After a node receives the ELECT messages from all the neighbors, it votes for the highest weight node by sending a VOTE message to this node.

If a node receives a VOTE message from each of its neighbors, it changes its color to RED. We call such a node as RED node. At the end of Phase I there are one or more RED nodes in the network, but they do not know each other.

3.3.2 Phase II

In Phase II, each RED node *r* starts a diffusing computation separately and unaware of the others. Each RED node then grows a tree rooted at it by sending a SEARCH message to all of its 1-hop neighbors. A neighbor of *r*, say node *i*, designates *r* as its predecessor, set *r* as its visitor node, and propagates the received SEARCH message to all its neighboring nodes (successors) except *r*. If a node has no successors to forward the SEARCH message received from its predecessor, it returns a SIGNAL to its predecessor embedding its node ID and node weight.

```
//This part is executed by each node i
(6) if (color_i = RED)
        send SEARCH(i) to each node j in nbr_i;
(7) when SEARCH(l) is received from node j
(8) if (visitor_i = NULL) {
        visitor_i ← l; pred_i ← j; insert i into the ResultQ_i;
        succ_i ←nbr_i \{pred_i}; send SEARCH(l) to nbr_k;}
(9) else {//assuming visitor_i =k
        if (k= l) send ACK() to j;
        else{
            send NACK(k, wt_k) to i; put (k, wt_k) into RedQ_i;
            send SIGNAL(RedQ_i, ResultQ_i) to pred_i;
        }}
(10) wait until an ACK or NACK or SIGNAL received from
                        each node j in succ_i{
        if (pred_i =0) GoTo Phase III;//this is the starting RED node
        else{
            merge RedQ_j into RedQ_i; merge ResultQ_j into ResultQ_i;
            send SIGNAL(RedQ_i, ResultQ_i) to pred_i;}
```

Fig. 2. Pseudocode of Phase II

If a WHITE node *j* receives a SEARCH message with root *r'* from some *k,* after *j* received a SEARCH message rooted from *r*, *j* includes *r'* in its $RedQ_j$ and sends a NACK to *k* telling *k* about *r*. On the reception of the NACK message, node *k* includes the information of *r* in its $RedQ_k$.

If a WHITE node receives a duplicate SEARCH message initiated by the same RED as its visitor node, it sends an ACK to the sender instantly.

When a node receives an ACK, a NACK or a SIGNAL from each of its successors, it sends a SIGNAL to its predecessor. This process is called the shrinking of the tree created by the starting RED node. When the RED node receives SIGNAL from all of its neighbors, then Phase II ends.

At the end of Phase II, every RED node knows one or more RED nodes and has visited a subset of the nodes in the network.

3.3.3 Phase III

After Phase II, if there is more than one RED node, each of them knows at least one other RED node. Then, in Phase III, the RED nodes exchange their partial knowledge of the weight values of nodes visited so that at the end, only the highest weight RED node receives the global knowledge about all the nodes in the environment. This RED node then selects the *K* highest weight nodes in the environment and sends them a LEADER message. The message exchange procedure is described as follows.

```
//This part is executed by each RED node i
(11) if (wt_i =min(RedQ_i) ){
        k← max(RedQ_i); P← RedQ_i\{k};
        send DIRECT(k, wt_k) to each node in P;
        send RESULT(|P|, ResultQ_i) to k;}
(12) else{
        P← RedQ_i\{k|wt_k≥wt_i};
        wait until a RESULT or DIRECT received from each j in P;
        for (a RESULT(Q, ResultQ_j) from j) {
            merge ResultQ_j into ResultQ_i; merge Q into RedQ_i;
            delete j from RedQ_i;}
        for (a DIRECT(k, wt_k) from j){
            put k into RedQ_i; delete j from RedQ_i;}
        if(|RedQ_i|>1){
            m←max(RedQ_i);
            send DIRECT(m, wt_m) to nodes in RedQ_i\{m};
            send RESULT(ResultQ_i) to m;}
        else{
            sort the nodes in ResultQ_i according to their weights;
            LDR ← top K of ResultQ_i;
            send a LEADER(LDR) message to nodes in LDR;}
    }
```

Fig. 3 Pseudocode of Phase III

Step 11. If a RED node *i* finds out that it is the least weight RED node in its $RedQ_i$, then it sends a DIRECT message to all the RED nodes in $RedQ_i$ directing them to send their RESULT to node *k*, the highest weight RED node in $RedQ_i$. Then, *i* sends its own RESULT to *k* with the set of nodes directed to *k* by *i*.

Step 12. If a RED node *i* finds out that its own weight is not the least in its $RedQ_i$, it waits to receive either a RESULT or a DIRECT from each lower weight RED node *j*. Upon the reception of a RESULT(*Q*, $ResultQ_j$) from *j*, node i will merge the $ResultQ_j$ and $RedQ_j$ into $ResultQ_i$ and $RedQ_i$ respectively. Also, node *i* deletes *j* from $RedQ_i$.

Upon the reception of a DIRECT(*k*, wt_k) from *j*, node *i* put *k* into $RedQ_i$ and deletes *j* from $RedQ_i$.

After receiving all the DIRECT/RESULT messages from lower weight nodes in $RedQ_i$, if there is some node in $RedQ_i$ with a higher weight than *i*, node *i* sends its $ResultQ_i$ to node *m*, the highest weight node in $RedQ_i$ and directs rest of the nodes by sending DIRECT(*m*, wt_m).

Otherwise, if *i* is the only node in $RedQ_i$, obviously *i* has the highest weight among all the nodes in the network. It sorts[1] $ResultQ_i$ and selects the top *K* nodes according to the weight values as leaders. Finally, node *i* sends the LEADER message to each leader node selected.

[1] For two nodes with the same weight value, the one with the lower node ID is viewed as a higher weighted node.

3.4 Handling node failures

In this subsection, we extend our algorithm to handle node failures. We consider node failure by crashing, i.e. permanent halting.

To handle failures, we assume a probing mechanism existing to enable a node to detect the failure of its neighbors. With the following modifications, our algorithm presented in Section 3.3 can handle node failures.

First, we consider the failure of RED nodes. We adopt the backup approach to handle such failures. After a node is elected as a RED node in Phase I, it will select the highest weight node among its neighbors as a backup node, called a GREEN node. The GREEN node will take over the role of RED node in case the later crashes. Such a mechanism is suitable and efficient considering that only few nodes are RED nodes and they are more reliable than other nodes.

Now, let us discuss the failure of a WHITE node. In Phase I, a node will delete a crashed neighbor node from its neighbor list, so as to avoid the block caused by the waiting for a VOTE message from a crashed node.

Then, we discuss the failure in Phase II. In case a failed node has empty child-set, it only requires sending ACK or NACK messages. The receiver node will detect the failure of this node and delete it from their neighbor set. If the crashed node *i* has a non-empty child set and a SEARCH message has been sent to some of them, child nodes can detect the failure of *i* and send their SIGNAL messages to the initiator of the SERRCH message, which is a RED node.

In Phase III, only RED nodes need to send and receive messages. Therefore, node failures of WHITE nodes in Phase III will not affect the algorithm at all. Of course, the node failures may change the network topology and affect the routing protocol, but this is out of the scope of our paper. Once the network is connected, Phase III can complete.

4. Performance evaluation

We have carried out simulations to evaluate the performance of our proposed algorithm. Two different versions of our proposed algorithm have been simulated. One is for fault-free scenarios and the other one is for scenarios with node failures. Moreover, to show the advantage of our algorithm, we also simulated a naive *K*-leader election algorithm, where each node initiates the diffusing computation to collect the weight information of all the nodes and elect the top *K* leaders based on the information collected.

4.1 Simulation setup and metrics

The simulation system consists of two modules: the network and the leader election algorithm. The main parameters of the simulations are shown in Table 1.

The network nodes are randomly scattered in a square territory. The total number of nodes is varied to examine the effect of system scale on the performance. Of course, to make the performance results in difference scenarios comparable, we also scale the territory size according to the total number of nodes.

For message routing, we implemented a simple protocol based on the "least hops" policy, which is adopted in many classical routing protocols in ad hoc networks. A routing table is proactively maintained at each node.

Table 1 Parameters of simulations

Number of Nodes, n	*20, 40, 60, 80, 100*
Territory Scale (m)	*280, 400, 500, 580, 650*
Transmission radius	*100m*
Routing-protocol	*Least hops*
Node failure rate	*15%*
K/n	*25%*

The leader election algorithms are implemented as applications run on top of the network. The weight values of the nodes are assigned randomly. In the scenarios with node failures, we set the percentage of faulty nodes to be 15%. The faulty nodes are randomly selected and a faulty node crashes in a randomly chosen time. The failure detection part is simulated based on the heartbeat-like approach.

In the simulations, we measure the performance of all the algorithms using the following metrics:

NM (*Number of Messages*): the total number of messages exchanged to elect the *K*-leader.. Here, a "message" refers to an "end-to-end" message, i.e. a message from the source node to the destination node. Such a message may be forwarded by several intermediate nodes in the network level.

NH (*Number of Hops*): the total number of hops of the messages exchanged to achieve the global decision. One "hop" means one network layer message, i.e. a point-to-point message. Compared with NM, NH can reflect the message cost of an algorithm more precisely.

Moreover, due to the false detection of failures, the fault-tolerant version of our proposed algorithm may not elect the top *K* leaders accurately, i.e. some elected leader may have crashed or may not have a weight ranked the top *K*. To examine the "accuracy" of the elected leaders, we define the metric "leader accuracy":

LA = k_s / *K*, where k_s is the number of elected nodes that do not crash and are top *K* weighted.

4.2 Simulation results

We discuss the simulation results according to the performance metrics used. For convenience, in the discussion of simulation results, our proposed algorithm and its fault tolerant version are denoted by "Prop" and "Prop-FT" respectively. The naive leader election algorithm is denoted by "Naive".

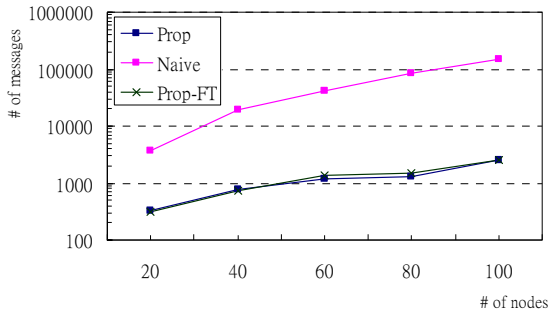

Fig. 4. Number of messages

4.2.1 Number of messages

Fig. 4 shows the number of messages of the different algorithms. Obviously, NM increases with the increase of number of nodes in the system. This is easy to understand. In a large system, more messages need to be exchanged. However, compared with Naive, the Prop and Prop-FT both need much fewer messages. This indicates our algorithm is message efficient. Moreover, Fig. 4 shows that the larger the system is, the more message cost is saved by our algorithm. The Naive algorithm has a message complexity of $O(n^2)$ while our proposed algorithm has a message complexity of $O(n \bullet d + n \bullet r)$, where d is the average number of neighbors of a node and r is the number of RED nodes in the network. Obviously, both d and r are much less than n, so the NM of Naive increases much faster than that of Prop and Prop-FT. Therefore, our proposed algorithm has very good scalability.

Comparing Prop and Prop-FT, we can see that, NM of Prop-FT is a little less than that of Prop. Due to node failures, not all the nodes participant in the leader election procedure, so Prop-FT needs a little few messages. Of course, the failure also causes the election results inaccurate as shown later.

4.2.2 Number of hops

Fig. 5 shows the results in NH. Since NH is generally determined by NM, the curves in Fig. 5 are similar to those in Fig. 4. NH increases with the increase of the system scale and our proposed algorithm costs fewer NH compared with Naive. However, compared with NM, the difference in NH is much larger. This is because that most messages of our algorithm appear in Phase I, and all these messages are one-hop messages. Therefore, the average number of hops per message is less than that of Naive.

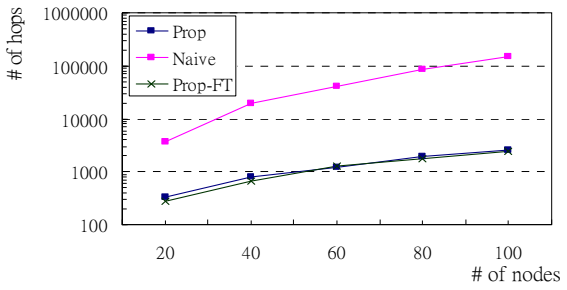

Fig. 5 Number of hops

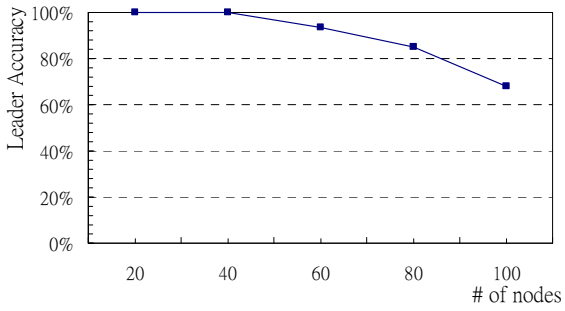

Fig. 6 Leader accuracy with node failures

4.2.3 Leader accuracy

We plot the LA under various numbers of nodes in Fig. 6. In general, the accuracy is not affected much by node failures. When the system is not large, we can achieve almost 100% accuracy. Considering the fault rate is 15%, such accuracy is satisfactory. LA decreases with the increase of system scale but it is still as high as 65% even in a system with 100 nodes. Therefore, our fault tolerance mechanism is effective and efficient.

5. Conclusions and future work

In this paper, we propose the first deterministic *K*-leader election algorithm. We consider a wireless ad hoc network where each node has a weight value that represents some performance property, e.g. processing capability, battery power. Our algorithm aims to elect the top *K* weighted nodes in the network. A diffusing computation approach is adopted to collect the weight information for electing leaders. To reduce message cost, we first locally choose some high weighted nodes, called RED nodes, to act as coordinators. Then, each RED node initiates a diffusing computation procedure to collect the weight information of other nodes collaboratively. Finally, the information collected by different RED nodes is merged together to elect the final leaders. We also design mechanism to handle node failures. The simulation results show that, benefiting from the use of RED nodes, our algorithm can elect leaders with much less message cost than a naive solution.

In future, we will extend our algorithm to handle node mobility. We are also planning to increase the fault tolerance capacity of our algorithm by considering more types of failures.

ACKNOWLEDGEMENTS

This research is partially supported by the Hong Kong RGC CERG grant PolyU 5105-05E and China National 973 Program Grant 2007CB307100

References

[1] M. Aguilera, C. Gallet, H. Fauconnier, S. Toueg. Stable leader election, LNCS 2180, p. 108 ff.

[2] J. Brunekreef, J. Katoen, R. Koymans and S. Mauw. Design and Analysis of Leader Election Protocols in Broadcast Networks, Distributed Computing, vol. 9 no. 4, pages 157-171, 1996.

[3] E.W. Dijkstra and C.S. Scholten. Termination Detection for Diffusing Computations, Information Processing Letters, vol. 11, no. 1, 1980.

[4] R. Ferreira, M. Ramanathan, A. Grama and S. Jagannathan, Randomized Protocols for Duplicate Elimination in Peer-to-Peer Storage Systems, IEEE Trans. Parallel Distrib. Syst. 18(5): 686-696 (2007).

[5] H. Garcia-Molian, Elections in a Distributed Computing System, IEEE Transactions on Computers, vol. C-31, no. 1, 1982.

[6] Kostas P. Hatzis, George P. Pentaris, Paul G. Spirakis, Vasilis T. Tampakas and Richard B. Tan, Fundamental Control Algorithms in Mobile Networks, Proc. 11th Annual ACM Symposium on Parallel Algorithms and Architectures, 1999.

[7] N. Malpani, J. L. Welch, and N. vaidya, Leader Election Algorithms for Mobile Ad Hoc Networks, Proc. of the 4th int'l workshop on Discrete algorithms and methods for mobile comp. and comm., Aug. 2000.

[8] K. Nakano and S. Olariu, Randomized Leader election protocols for ad hoc networks, Proc. of SIROCCO, 2000.

[9] Sung-Hoon Park, An Election Protocol in a Mobile Environment, Proc. of PDPTA2000, pp. 200-210, June 2000.

[10] V. D. Park, and M. S. Corson. A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks, Proc. IEEE INFOCOM, Apr 1997.

[11] V. Sundramoorthy, and J. Scholten, Ir. and P.G.Jansen, Ir. and P.H. Hartel, Service discovery at home, Proc. of Joint Conf. of the 4th Int'l Conf. on Info., Comm. and Signal Processing, and the 4th Pacific Rim Conf. on Multimedia, 2003.

[12] G. Taubenfeld. Leader Election in presence of n-1 initial failures, Information Processing Letters, vol.33, no.1, pages 25-28, October 1989.

[13] S. Vasudevan, J. Kurose and D. Towsley, Design and Analysis of a Leader Election Algorithms for Mobile Ad Hoc Networks, Proc. of ICNP04, pp. 350–360.