Performance analysis of Leader Election Algorithms in Mobile Ad hoc Networks

Muhammad Mizanur Rahman †, M. Abdullah-Al-Wadud ††, Oksam Chae††

Dept. of Computer Science & Information Technology, Islamic University of Technology,
Board Bazar, Gazipur-1704, Bangladesh†
Dept. of Computer Engineering, Kyung Hee University, 1 Seocheon, Kiheung, Yongin, Gyonggi, Korea††

Summary
Leader election is an extensively studied problem in Ad hoc networks. In our study, we have implemented an extended idea of an existing leader election algorithm for energy saving to arbitrary topological changes. In this method, our focus is to reduce the number of leader election processes; to make it more energy efficient. Unlike the previous solutions, the algorithm proposes that each node maintains a list of candidates to minimize the total number of leader elections. Simulation results show that this leader election algorithm using candidates has much fewer leader elections and generates lower messages than the existing leader election algorithm.

Key words:
Leader Election Algorithm for Ad hoc Networks (LEAA), Candidate Based Leader Election Algorithm for Ad hoc Networks (CBLEAA), heart-beat-message, most-valued-node, computation-index.

1. Introduction

A mobile Ad hoc network (MANET) is a collection of mobile nodes that can communicate via message passing over wireless links. Each node communicates directly with other nodes within a specified transmission range. The nodes communicate via other nodes if they are not within a specified transmission range.
Leader election is a fundamental control problem in both wired and wireless systems (e.g. MANET, Sensor networks). For example, in group-communication protocols, a leader election is necessary when a group leader crashes or departs from the system [1]. Leader election has a large number of applications such as key distribution [4], routing coordination [5], sensor coordination [6] and general control [3, 7]. It can serve for creating particular tree communication structures [2] and other standard problems in distributed systems.
The algorithm of the leader election problem [8] elects a unique leader from a fixed set of nodes. To accommodate frequent topology changes, leader election in MANET has to be adaptive. The elected leader should be the most-valued-node among all the nodes of the network. The value for the leader node selection is a performance-related characteristic such as remaining battery life, minimum average distance from other nodes or computation capabilities [1].
Many solutions are proposed for leader election, but most of them are not fit perfectly to dynamic nature of mobile networks. Leader election algorithm provided in [7], maintains a directed acyclic graph with a single sink, which is the leader. However this algorithm [7] is proved correct for a completely synchronous system with a single link change. The algorithms proposed in [9, 10] work only if the topology remains static and hence cannot be used in mobile networks. Self-stabilizing spanning tree algorithms [12, 13] assume a shared-memory model and are not suitable for an Ad hoc network. Besides, several clustering and hierarchy-construction schemes [11, 14] may be adopted to perform leader election, but they cannot be used in an asynchronous mobile system. Leader election algorithm presented in [1], manages the election process very efficiently. But the method requires a large number of leader elections, which is not very supportive to energy conservation. To solve this problem, we implement an algorithm which is based on Leader election algorithm presented in [1], but every node keeps a leader list instead of a single leader. This algorithm is mainly emphasized for lower power consumption.
The rest of the paper is organized as follows: section 2 describes the existing leader election algorithm LEAA while section 3 presents the leader election algorithm using candidates CBLEAA. In section 4, comparative performance analysis between the leader election algorithms is done through simulation. Finally, section 5 discusses about our future work and conclusions.

2. Existing algorithms for Leader Election

In this section, we describe an existing leader election algorithm LEAA [1], which is based on classical

Manuscript received February 5, 2008
Manuscript revised February 20, 2008

termination detection algorithm for diffusing computations by Dijkstra and Scholten [15]. First we describe the algorithm for static networks and then for mobile environments.

2.1 Leader Election in a Static Network

This algorithm acts under the assumption that nodes and links never fails. It uses three types of messages, viz. Election, Ack and Leader. The algorithm works as follows:
Election: If a leader node doesn't exist in the network, an initiator node transmits Election message to the immediate neighbor nodes. The neighbor nodes propagate the messages to their neighbors. This process is continued until all leaf nodes get the Election messages. This phase is referred as the growing phase of the spanning tree.

Ack: When any node receives an Election message from a neighbor (not parent), it immediately responds with an Ack message. Instead of sending Ack message to its parent, a node waits until it receives Acks from all its children. On receipt of the Election message, every leaf node sends an Ack message along with its own ID, to its parent. The parent node compares its own ID with these incoming IDs from all its children. Then it selects the highest one and sends it through the Ack message to its parent. This process is continued until the initiator node gets all Acks from all children. This phase is referred as the shrinking phase of the spanning tree.

Leader: When the initiator node gets Ack messages from all its children, it selects the highest ID as the leader node. It then broadcasts this ID in the Leader message to all nodes of the network.

Figure 1 shows an example of such leader election. In figure 1(a), node 3 is the initiator that sends Election (E) message to its neighbor. In figure 1(b), nodes 2 and 5 set their pointers to point to parent node 3. They get Election messages from each other and immediately acknowledged. Immediate acknowledgements are not shown in the figure. In figure 1(c), a complete spanning tree is created. In figure 1(d), nodes 7 and 9 send their Ack messages (A) to their parents with their own IDs. In figure 1(e), nodes 2 and 5 compare their own IDs with the incoming ones and send the higher IDs in Acks to node 3. In figure 1(f), node 3 selects 9 as the leader ID and broadcasts it via the Leader message (L).

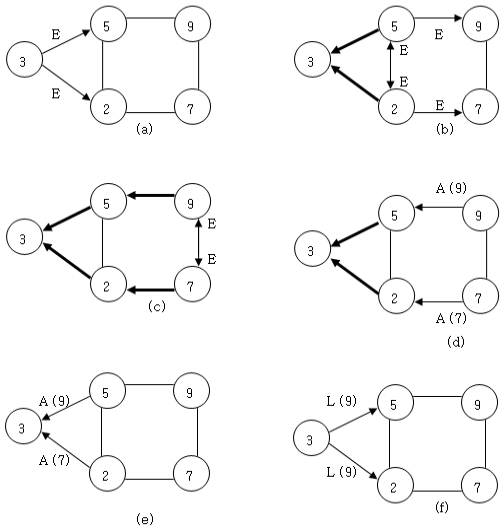


Fig. 1 An execution of leader election algorithm based on Dijkstra-Scholten termination detection algorithm.

2.2 Leader election in mobile environment

In this section we briefly describe LEAA. If the leader node crashes or goes out of the network, a new leader election is necessary for that network.

The leader node of a connected network periodically (after each 20 seconds) sends heart-beat-messages to other nodes. The absence of heart-beat-messages from its leader for a predefined timeout period (6 times) triggers a fresh leader election process at a node. Then the election proceeds as mentioned in the previous section. But when node mobility, node crashes, link failures, network partitions and merging of partitions are introduced during the leader election process, that simple algorithm is not sufficient. To solve this problem, two extra messages, Probe and Reply are used.

In the dynamic environment, a number of events may occur that may change the topology of a MANET. To handle such situations, LEAA applies the following techniques in the leader election process.

Handling Multiple, Concurrent Computations: Here more than one node may concurrently detect leader departure. These nodes initiate separate leader election independently that leads to concurrent leader elections. To handle this situation, the algorithm LEAA requires each node participates in only one diffusing computation at a time. To achieve this each diffusing computation is identified by a computation-index. This computation-index is a pair, viz. num, ID, where ID represents the

identifier of the node that initiates this computation and num is an integer as described below.
$\langle num_1, ID_1\rangle > \langle num_2, ID_2\rangle \Leftrightarrow ((num_1 > num_2) \vee ((num_1 = num_2) \wedge (ID_1 > ID_2)))$

A leader election with higher computation-index has higher priority than another leader election. When a node participates in a leader election, hears another leader election with a higher computation-index, stops its current execution. Eventually a node with highest computation-index initiates the leader election process.

Handling Network Partitions: Once a node joins in a leader election, it must receive Ack messages from all of its children, before it sends the Ack message to its parent. However, during the shrinking phase of the spanning tree, some nodes may go out of the network. To detect such events, each node sends periodic Probe messages to the neighbors of the spanning tree. A node which receives the Probe message, responds with a Reply message. If a node fails to get Reply message from a node for a certain timeout period, removes that node from its neighbor list of the spanning tree. A node must detect this event; otherwise it never reports an Ack to its parent. Eventually the leader node cannot be determined.

Handling Partition Merges: Node mobility can merge network partitions, when at least two nodes from different partitions come in the communication range of each other. In this case both nodes exchange each other's leader information. The node having lower leader ID accepts the other leader as the new leader of its partition and propagates the message to other nodes of that partition.

Node Crashes and Restarts: If a node failure creates network partitions, appropriate actions are taken as described earlier. When a node recovers from a crash, becomes a node without leader and so starts a new election to find its leader.

3. Proposed Algorithm

This algorithm is based on LEAA, which is described in the previous section. In this algorithm, we propose a list of leaders instead of just one leader to be maintained in every node. Here we want to take the advantage of having multiple candidates. Each node contains a leader list of five nodes (in descending order) as the threshold value, where the first node is considered as the active leader of the network. If the first one is absent for a specified period, the second becomes the active leader and so on. If we consider more candidates than this threshold value, there is a possibility of message overhead, which consumes a lot of energy.
We follow the assumptions and constraints that are mentioned in [1]. We assume MANET is an undirected graph. Here vertices represent the mobile node and an edge represents the communication link between any two nodes within communication range. Thin arrows represent the direction of flow of messages and thick arrows indicate pointer of child node to parent. In this algorithm, we apply the following constraints:

- All nodes have unique identifiers
- We consider the node's ID (identifier) as the key value for leader election (for simplicity to descri be and simulate), i.e. the node having the highest ID in a network is considered as the *most-value d-node* (leader).
- Links are bidirectional and FIFO (First in First o ut).
- Node mobility can change topology arbitrarily, i ncluding network portioning/merging. Furtherm ore, node can crash and come back to its original network at any time.
- Each node has a large buffer to avoid buffer ove rflow at any point in its lifetime.

3.1 Candidate based leader election algorithm (CBLEAA)

We implement a candidate based leader election algorithm based on LEAA. Here every node maintains Leader list (L) and Set the size of L to 5 (Threshold value). Empty ID is denoted by -1 in L.

1. Check the existence of the leader in the network, after the *heart-beat-message* interval (20 seconds).
 1.1 If Leader doesn't exist in the network after the specified period (Six times *heart-beat-message* interval)
  (a) Select an initiator node
  (b) Initiator node sends the *Election* message to all of its children. This process is continued until these messages reach to all leaf nodes.
  (c) For the leaf nodes, l, in the network
   I. Add l to its own Leader list, L.
   II. Send L to its parent within *Ack* message.
   III. Parent node sorts (in descending order) i ts ID with the contents of L. The node pr opagates L to its parent. Repeat this proc ess until the initiator node gets all the Le ader lists from each branch.
   IV. In this shrinking phase of the spanning tr ee, each node sends periodic *Probe* mess age and waits for *Reply* message from th e neighbors, to maintain the connectivity

among the nodes.

(d) Initiator node selects the *highest-valued-L* from the collected Leader lists and send this

to all the nodes of the network, within *Leader* message.

1.2 If at least one leader exists in *k*th (position in *L*) level of the Leader list

(a) The corresponding leader node broadcasts *heart-beat-message.*

(b) Update Leader list by setting the active leader of kth level of *L* in the first position by shifting the invalid leaders of *L* for future rejoin operation.

1.3 If several active leaders of different networks, exist in the same network

(a) Multiple Leaders lists from previous networks combine and select new five candidate nodes by keeping the active leaders in sequence (in descending order) in the front positions of the list. After that the first positioned node of the list becomes the active leader node.

3.2 Leader election in mobile environment

We now describe how this algorithm accommodates arbitrary changes in topology introduced by node mobility. Our algorithm shares the idea of multiple concurrent computation, network partitioning and merging, node crashes and restarts during the leader election process, provided by LEAA.

Leader election process: Figure 2(a) to 2(c) show the growing phases of the spanning tree. Every node maintains a Leader list. In our example we use the size of the Leader list as 5. Figure 2(d) shows that the leaf nodes 7 and 9 add their IDs in the list and send these to their parents in the *Ack* messages. Figure 2(e) shows that the initiator receives two lists from its two branches and they are A (7, 2, -1, -1,-1) and A (9, 5, -1, -1,-1). From these two lists, the initiator node selects the Leader list L (9, 7, 5, 3, 2) and broadcasts to all nodes of the network. Like LEAA, here all nodes send periodic *Probe* messages and wait for the *Reply* from the neighbors of the spanning tree, to maintain the connectivity.

Unlike LEAA, in the following section, we will see how this algorithm manages network partitions and merges other than the leader election process time.

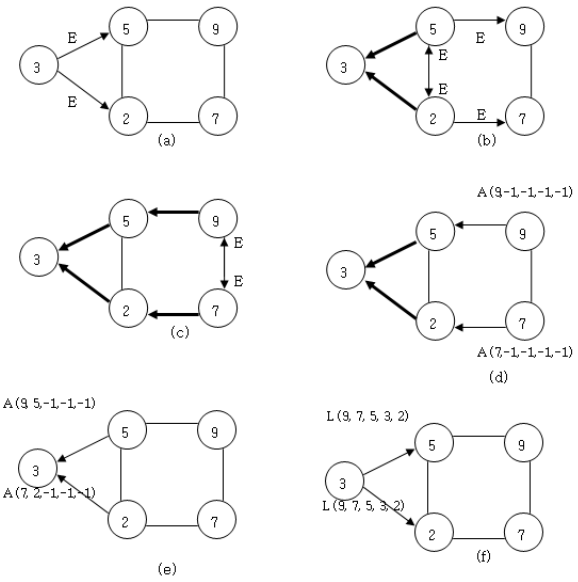


Fig. 2 The leader election process by CBLEAA

Handling Network Partitions: In figure 3, we see the advantage of having multiple candidates. In figure 3(a), all nodes of the network maintain the same Leader list, where the active leader is 50. But as node 3 disappears, two networks are created. In figure 3(b), on the right network, nodes 50, 20, 17 exist. So this network does not modify the Leader list. But on the left network, the first three candidate IDs are absent in the leader list. So after the time out of three levels of leader nodes checking, node 10 sends the heart-beat-message to all nodes of the network. So unlike LEAA, a new leader election is not necessary here. To reduce the waiting time, all nodes update their Leader lists by shifting the invalid IDs to the end of the Leader lists. But we shouldn't delete these nodes from the Leader lists for a while, for future rejoin.

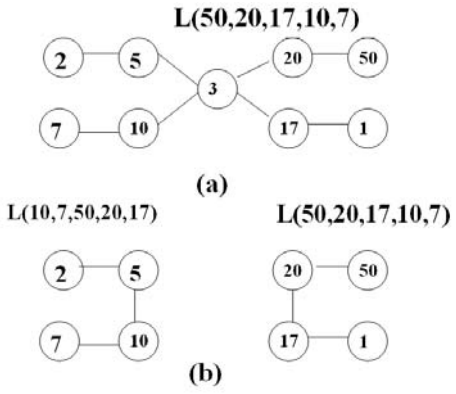


Fig. 3 Handling network partitions by CBLEAA.

Handling Partition Merges: Network merging can be managed efficiently by this method. In figure 4(a), there are three networks and they maintain the Leader lists L (10, 7, 5, 2, and 1), L (25, -1, -1, -1, -1) and L (50, 20, 17, 1, -1). According to our algorithm every node in the merged network has finally L (50, 25, 10, 20, 17), where nodes 50, 25 and 10 were the active leaders. After the merging operation node 50 becomes the active leader. This process is shown in Figure 4 (b).

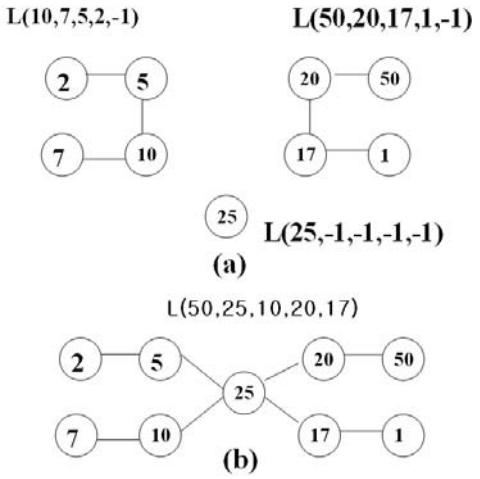
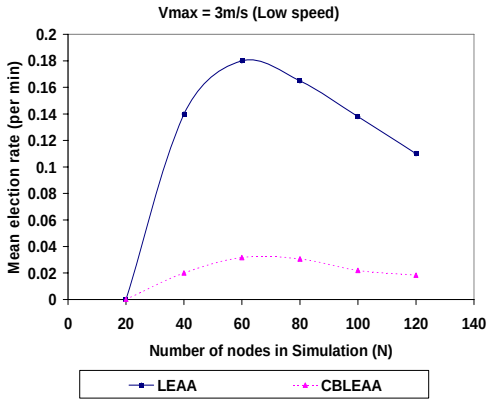


Fig. 4 Handling partition merges by CBLEAA.

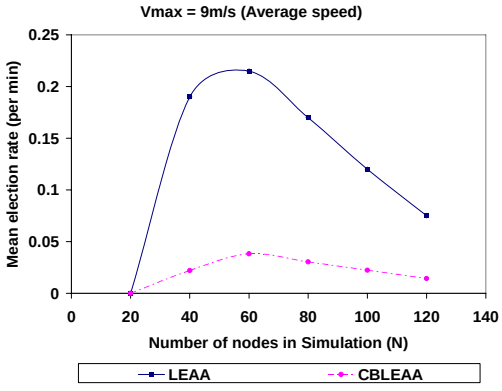
4. Simulation result

We compare the performance between LEAA and CBLEAA through simulations. Both leader election algorithms are implemented in C++. Here MANET size is 2000*2000 square meters. In the simulation, nodes can move from 1 m/sec (Vmin) to maximum 19 m/sec (Vmax). To see the effect of transmission range over the algorithms, we use transmission ranges of 200, 250 and 300 meters. We set message traversal time between the two nodes to 0.03 second as default value. We allow the number of nodes (N) up to 120 in the simulation area. Due to the node mobility, several nodes can go out of the simulation area and can enter into the simulation area at any time. As the requirement of the existing algorithm, each simulation runs for the duration of 100 minutes. Finally the simulation results are taken from the averaged values of 20 simulations run times.

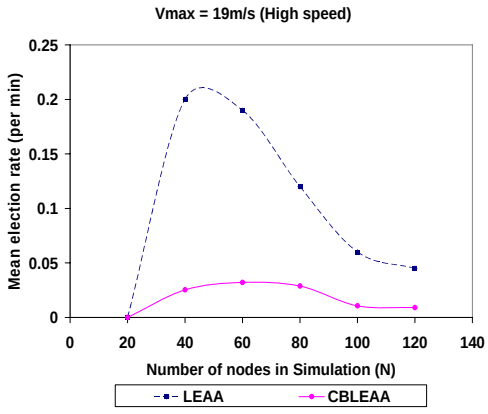
Impact on Node Density: In order to study the impact of node density, we vary the maximum node speed Vmax. The graphs in figure 5 show the Election Rate for three different values of Vmax viz. 3m/s, 9m/s and 19m/s. In these graphs, we see the Election Rate of node first increases with node density (N), and then start decreasing with any further increase in N. This is because when N = 20, most of nodes are expected to be isolated. But as N increases, there are few networks with few nodes. Node mobility causes frequent leader departures and hence Election Rate increases. But after a certain threshold, the node density becomes very high and most of the nodes belong to large networks. As networks remain connected for long duration, Election Rate drops. From figure 5, we see the candidate based method has much lower Election Rate than that of the existing method.



(a)



(b)



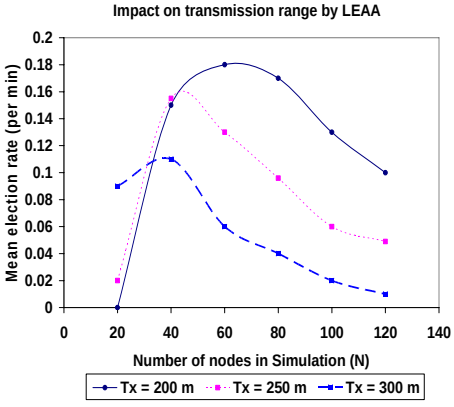
(c)

Fig. 5 Average Election Rate Vs V_{max}. Here V_{min} = 1 m/sec and transmission range of each node is 250 meter.

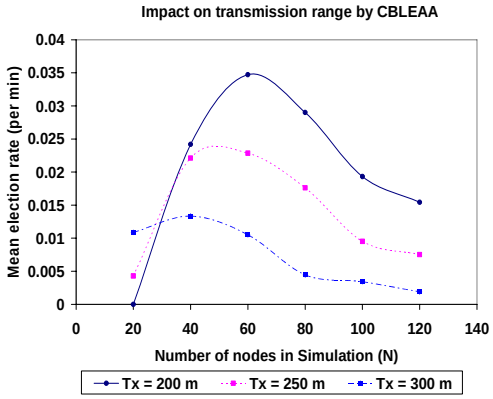
Impact on transmission range: We study the impact of Transmission range (T_x) on Election Rate for three different choices of T_x, viz. 200m, 250m and 300m. From the graphs of figure 6, we see increased transmission range of nodes leads to a higher Election Rate when N is small (i.e. N = 20). This is because, for large value of T_x, there are fewer isolated nodes, but each network has few nodes. Due to node mobility here Election Rate is higher. But for large values of N, the Election Rate becomes smaller with increase in T_x. The reason is the network sizes are larger for large values of T_x and partition occurs less frequently. As before, in these graphs, we see that CBLEAA has significant performance advantage over LEAA. Here we see for all values of T_x, Election Rate by the candidate based method is much smaller than that of the existing method. These graphs also show that large transmission range is ideal for leader election methods.

5. Conclusions and future work

Energy saving is an important research area for Ad hoc and Sensor networks. To achieve this purpose, we tried to derive leader election algorithm, which successfully guarantees that every node must have leader in any situation and save energy in mobile Ad hoc networks. For saving energy our focus is to reduce the number of leader elections processes. Because leader election needs three phases of transmissions and receptions of messages that use a lot of energy. Our simulation results show that the candidate based algorithm (CBLEAA) has significant energy saving feature than that of other traditional algorithms. Another important fact is that energy effective candidate based algorithm is particularly simple and straight for understanding. Our future plan is to evaluate the run time complexity of this algorithm. We have also plan to implement the practical protocol.



(a)



(b)

Fig. 6 Average Election Rate Vs T_x. Here V_{min} = 1 m/sec and V_{max} = 3m/s.

References

[1] Vasudevan, S., Kurose, J, Towsley, D. Design and Analysis of a Leader Election Algorithm for Mobile Ad Hoc Networks. Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP) (2004) 350-360

[2] Y. Afek and A. Bremler. Self-stabilizing unidirectional network algorithms by power supply. Chicago Journal of Theoretical Computer Science, December 1998.

[3] O. Bayazit, J. Lien, and N. Amato. Better group behaviors in complex environments using global roadmaps. 8th International Conference on the Simulation and Synthesis of living systems (Alife '02), Sydney, NSW, Australia, pp. 362-370, December 2002.

[4] B. DeCleene et al. Secure group communication for Wireless Networks. In proceedings of MILCOM 2001, VA, October 2001

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

[6] W. Heinzelman, A. Chandrakasan and H. Balakrishnan. Energy-Efficient Communication Protocol for Wireless Micro sensor networks. In proceedings of Hawaiian International Conference on Systems Science, January 2000.
[7] N. Malpani, J. Welch and N. Vaidya. Leader election Algorithms for Mobile Ad Hoc Networks. In fourth International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, Boston, MA, August 2000.
[8] N. Lynch. Distributed Algorithms. 1996, Morgan Kaufmann Publishers, Inc.
[9] R. Gallager, P. Humblet and P. Humblet and P. Spira. A Distributed Algorithm for Minimum Weight Spanning Trees. In ACM Transactions on Programming Languages and Systems, vol.4, no.1, pages 66-77, January 1983.
[10] D. Peleg. Time Optimal Leader Election in General Networks . In journal of Parallel and Distributed Computing, vol.8, no.1, pages 96-99, January 1990.
[11] D. Coore, R. Nagpal and R. Weiss. Paradigms for Structure in an Amorphous Computer. Technical report 1614, Massachussetts Institute of Technology Artificial Intelligence Laboratory, October 1997.
[12] Y. Afek, S. Kutten and M.Yung. Local Detection for Global Self Stabilization. In Theoretical Computer Science, Vol 186, No. 1-2, 339 pp. 199-230, October 1997.
[13] S. Dolev, A. Israeli and S. Moran. Uniform dynamic self-stabilizing leader election part 1: Complete graph protocols. Preliminary version appeared in proceedings of 6th International Workshop on Distributed Algorithms (S. Toueg et.al., eds.), LNCS 579, 167-180,1992),1993.
[14] C. Lin and M. Gerla. Adaptive Clustering for Mobile Wireless Networks. In IEEE journal on selected areas in communications,15(7): 1265-75, Sep 1997.
[15] E.W. Dijkstra and C.S. Scholten. Termination detection for diffusing computations. In Information Processing Letters, vol. 11, no. 1, pp. 1-4, August 1980.

Muhammad Mizanur Rahman received his B.Sc. (Hons) degree in computer science from the University of Dhaka, Bangladesh in 2003 and MS in Computer Engineering from Yeungnam University in 2006. In 2004, he joined as a lecturer in the Faculty of Computer Science and Engineering, University of Development Alternative, Bangladesh.. In 2006, he joined as a Faculty member in Computer Science and Information Technology, Islamic University of Technology, Bangladesh. His research interest includes Wireless Networking, Digital image processing.

M. Abdullah-Al-Wadud received his B.S. degree in computer science and M.S. in computer science and engineering from the University of Dhaka, Bangladesh in 2003 and 2004, respectively. In 2003, he joined as a lecturer in Faculty of Sciences and Information Technology, Daffodil International University, Bangladesh. In 2004, he joined as a lecturer in Faculty of Sciences and Engineering, East West University, Bangladesh. Currently he is pursuing his PhD degree in Department of Computer Engineering, Kyung Hee University, South Korea. His research interest includes image enhancement, medical image processing, and pattern recognition.

Oksam Chae received his B.S degree in electronics engineering from Inha University, South Korea in 1977. He completed his MS and PhD degree in electrical and computer engineering from Oklahoma State University, USA in 1982 and 1986 respectively. During 1986-88 he worked as research engineer in Texas Instruments Image Processing Lab, USA. From 1988, he is working as a professor in Department of Computer Engineering, Kyung Hee University, South Korea. His research interest includes multimedia data processing environment, intrusion detection system, sensor networks and medical image processing in dentistry. He is a member of IEEE, SPIE, KES and IEICE.