A Distributed Backbone Formation Algorithm for Mobile Ad Hoc Networks

Orhan Dagdeviren and Kayhan Erciyes

Izmir Institute of Technology
Computer Eng. Dept., Urla, Izmir 35340, Turkey
{orhandagdeviren, kayhanerciyes}@iyte.edu.tr

Abstract. Construction of a backbone architecture is an important issue in mobile ad hoc networks(MANET)s to ease routing and resource management. We propose a new fully distributed algorithm for backbone formation in MANETs that constructs a directed ring architecture. We show the operation of the algorithm, analyze its message complexity and provide results in the simulation environment of ns2. Our results conform that the algorithm is scalable in terms of its running time and round-trip delay against mobility, surface area, number of nodes and number of clusterheads.

1 Introduction

MANETs do not have any fixed infrastructure and consist of wireless mobile nodes that perform various data communication tasks. MANETs have potential applications in rescue operations, mobile conferences, battlefield communications etc. Clustering has become an important approach to manage MANETs. In large, dynamic ad hoc networks, it is very hard to construct an efficient network topology. By clustering the entire network, one can decrease the size of the problem into small sized clusters. Clustering schemes can be classified as Dominating Set(DS)-based, low-maintanence, mobility-aware, energy-efficient, load-balancing and combined-metrics-based clustering [1]. DS-based clustering algorithms [2,3,4,5,6] like Wu's CDS(Connected Dominating Set) algorithm [2], Chen's WCDS(Weakly Connected Dominating Set) algorithm [3], Dominating Set Based Clustering Algorithm [4] try to find a DS for a MANET so that the number of mobile nodes that participate in route search can be reduced. Low-maintenance clustering [7,8,9,10] schemes aim at providing stable cluster architectures for upper-layer protocols with little cluster maintenance costs. Mobility-aware clustering [11,12,13] takes the mobility behavior of mobile nodes into consideration. Energy-efficient clustering [14,15,16] manages to use the battery energy of mobile nodes wisely in a MANET. Load-balancing clustering schemes [14,17,18] attempt to limit the number of mobile nodes in each cluster to a specified range so that clusters are of similar size. Combined-metrics-based clustering [19] usually considers multiple metrics, such as node degree, cluster size, mobility speed and battery energy in cluster configuration, especially in clusterhead decision [1].

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 219–230, 2006.

Load-balancing clustering schemes like Merging Clustering Algorithm(MCA) [17], Adaptive Multi-hop Clustering [18] (AMC) and Degree-Load-Balancing Clustering (DBLC) [14] distribute the workload of a network more evenly into clusters by limiting the number of mobile nodes in each cluster in a defined range. But the weakness of these algorithms is the lack of virtual backbone formation to serve the lower layer protocols like routing, or the upper layer operating system services like distributed *mutual exclusion* protocol [20]. In this study, we propose a backbone formation algorithm for load-balancing clustering algorithms where backbone is constructed as a ring architecture by directing clusterheads in a minimum spanning tree to each other. Related work in this area is reviewed in Section 2, we define, illustrate and analyze our algorithm in Section 3, provide implementation results in Section 4 and the final section provides the conclusions drawn.

2 Background

MCA finds clusters in a MANET by merging the clusters to form higher level clusters as mentioned in Gallagher, Humblet, Spira's algorithm [21]. The clustering operation we apply, however, operates by discarding the minimum spanning tree. This reduces the message complexity from *O(nlogn)* to *O(n)*. Upper and lower bound heuristics for clustering operation are used which result in a balanced number of nodes in the cluster formed. AMC maintains multihop cluster structure as similar to MCA. For cluster maintenance, each mobile node periodically broadcasts its information, its id, cluster id and status to others within the same cluster. Clusters are obtained by merging, and upper and lower bounds are used for controlling the cluster size. DLBC periodically runs the clustering scheme in order to keep the number of nodes in each cluster approximately equal to a system parameter, *ED*, which indicates the optimum number of mobile nodes that a clusterhead can handle. A clusterhead degrades to an ordinary member node if the difference between *ED* and the number of mobile nodes that if currently serves exceeds some value, *Max_Delta* [1]. As mentioned, load-balancing algorithms partition the network into a balanced number of clusters but a backbone is not constructed.

Wu et al.'s CDS Algorithm is a step wise operational distributed algorithm, in which every node has to wait for others in a lock state. In this algorithm, nodes exchange neighbor list messages to decide marking process. Algorithm has two phases of marking operation to find a connected dominating set. A CDS with small size reduces the number of nodes involved in routing-related tasks. Further heuristics and degree checking functionalities are added in Dominating Set based Clustering Algorithm to find the minimal CDS. The number of clusters produced by the CDS clustering is rather large and the cluster structure is highly overlapping [1]. Chen proposed a WCDS scheme by relaxing the requirement of direct connection between neighboring dominating nodes. Backbone formation is supported by the construction of CDS or WCDS in these algorithms, but adjusting the cluster size is not mentioned.

3 Our Algorithm

3.1 General Idea of the Algorithm

The algorithm we propose constructs a backbone architecture on a clustered MANET. Different than other algorithms, the backbone is constructed as a directed ring architecture to gain the advantage of this topology and to give better services to other middleware protocols such as *distributed mutual exclusion* [20] and *total order multicast*. The second contribution is to connect the clusterheads of a balanced clustering scheme which completes two essential needs of clustering by having balanced clusters and minimized routing delay. Besides these, the backbone formation algorithm is fault tolerant as the third contribution.

3.2 Description of the Algorithm

We assume that the MANET is partitioned by a load-balanced clustering algorithm like MCA, AMC or DLBC. Each node has distinct *node_id*, knows its *clusterhead_id* are the basic assumptions of our algorithm as well as these clustering algorithms.

Our main idea is to maintain a directed ring architecture by constructing a minimum spanning tree between clusterheads and classifying clusterheads into *BACKBONE* or *LEAF* nodes, periodically. To maintain these structures, each clusterhead broadcasts a *Leader_Info* message by flooding. In this phase, clustermember nodes act as routers to transmit *Leader_Info* messages. Algorithm has two modes of operation; hop-based backbone formation scheme and position-based backbone formation scheme. In hop-based backbone formation scheme, minimum number of hops between clusterheads are taken into consideration in a minimum spanning tree construction. Minimum hop counts can be obtained during flooding scheme. For highly mobile scenarios, an agreement between clusterheads must be maintained to guarantee the consistent hop information. In position-based backbone formation scheme, positions of clusterheads are used to construct the minimum spanning tree. If each node knows its velocity and the direction of the velocity, these information can be appended with a timestamp to the *Leader_Info* message to construct a better minimum spanning tree. But in this mode, nodes must be equipped with a position tracker like a GPS receiver.

Every node in the network performs the same local algorithm. The finite state machine of the algorithm is shown in Fig. 1. Each node can be either in *IDLE*, *BACKBONE* or *LEAF* states described below.

- *IDLE*: Initially all clusterheads are in *IDLE* state. If *Period_TOUT* occurs, each clusterhead broadcasts a *Leader_Info* message to the destination node and will make a state transition to *WT_INFO* state. If *Leader_Info* message is received, the clusterhead makes a state transition to *LEAF* state and reconstructs the ring by reorganizing the minimum spanning tree.
- *WT_INFO*: A clusterhead in *WT_INFO* state waits for *Leader_Info* message. If a *Leader_Info* message is received, the clusterhead makes a state

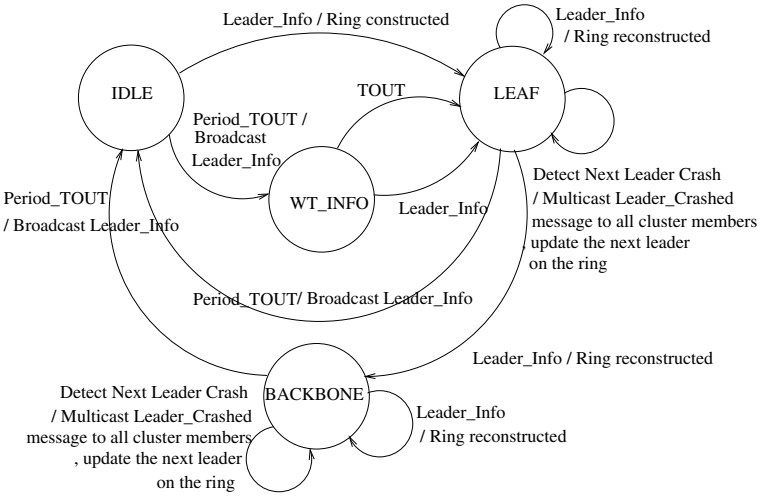

Fig. 1. Finite State Machine

transition to *LEAF* state and reconstructs the ring. If *TOUT* occurs, clusterhead makes a transition to *LEAF* state which indicates that the network has only two active partitions.

- *LEAF*: A clusterhead in *LEAF* state has a degree of 1 in its local minimum spanning tree. If a *Leader_Info* message is received, the clusterhead reconstructs the ring and makes a state transition to *BACKBONE* state if the degree exceeds 1. If *Period_TOUT* occurs, clusterhead makes a transition to *IDLE* state to restart the backbone formation.
- *BACKBONE*: A clusterhead in *BACKBONE* state has a degree greater than 1. For each *Leader_Info* message received, the ring is reconstructed. If *Period_TOUT* occurs, the backbone formation is restarted.

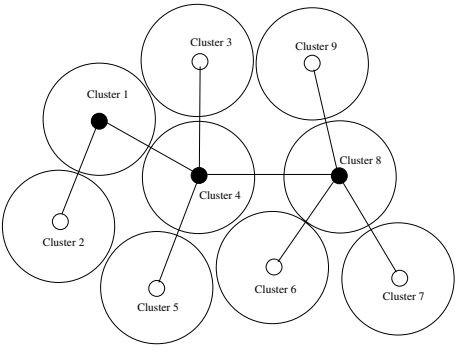

Fig. 2. MANET with its minimum spanning tree

```
1.Procedure ring_construct
2.begin
3.  construct minimum spanning tree by total received leader information
4.  if my degree is equal to 1
       execute ordinary_leaf
9.  else
10.     set my state to BACKBONE
    if I am a BACKBONE leader or a LEAF leader which can't find next leader
        execute backbone_proc
15.end
```

Fig. 3. Procedure executed by all leaders to construct a Ring Architecture

A balanced clustered MANET with its clusterheads and minimum spanning tree is shown in Fig. 2. *BACKBONE* clusterheads are shown as black and *LEAF* clusterheads are shown as white nodes. The main part of the algorithm is the construction of a ring architecture by orienting clusterheads in the minimum spanning tree. General idea is to divide the ring into two parts. A directed path of *BACKBONE* clusterheads and a directed path of *LEAF* nodes. Finally, these two directed paths are connected to each other to maintain the ring architecture. Each clusterhead aims to find the next clusterhead(leader) to construct the ring architecture by the procedure in Fig. 3.

Our first aim is to form the vital part of the backbone. The *BACKBONE* clusterheads are directed to each other from starting *BACKBONE* clusterhead to the end. Starting *BACKBONE* clusterhead is the one with the smallest connectivity to other *BACKBONE* nodes. This selection policy of *BACKBONE* clusterhead results in smaller hops and reduced routing delay. Ending *BACKBONE* clusterhead is directed to its *LEAF* with the smallest *node_id*.

LEAF leaders firstly execute the procedure in Fig. 5 to find the next leader on the ring. The aim of directing *LEAF* leaders with the same *BACKBONE* leaders to each other is to make the routing process over the same *BACKBONE* leader to reduce delay. *LEAF* leaders which can't find the next leader execute the procedure in Fig. 4 and search for a *LEAF* leader from the previous BACKBONE leaders of their parent to find a *LEAF* leader. Our last aim is to connect the *LEAF* leaders of different *BACKBONE* parents to maintain the routing operation by using the *BACKBONE* leaders.

Third contribution of our algorithm is the fault tolerance of clusterheads. Each clusterhead can maintain the list of cluster member nodes in load-balancing algorithms like MCA, AMC or DLBC. In our backbone formation algorithm, this list can be appended to *Leader_Info* message by each clusterhead. After the formation of the ring is completed, if a clusterhead detects the crash of the next clusterhead, it can multicast a *Leader_dead* message to all cluster members which initiates clustering operation. To support this functionality, clustering layer must be updated. If this crash occurs during a real time operation,

```
1.Procedure backbone_proc
2.begin
3.      find the starting BACKBONE leader such that its connectivity to
        other BACKBONE nodes is smallest between all other BACKBONE
        leaders.
4.      find the next leader of starting BACKBONE.
5.      If next leader found
6.        set the temporary BACKBONE leader to next leader of starting
          BACKBONE.
7.      If not found
8.        find LEAF leader with smallest node_id of starting BACKBONE leader.
9.       mark the starting BACKBONE leader.
10.     if I am starting BACKBONE leader set my next leader to found
        value
11.     else
12.         while all BACKBONE nodes are not marked
13.           find the next BACKBONE leader of temporary BACKBONE leader
              with smallest distance which is not marked.
14.           if found
15.              set the next leader of temporary BACKBONE leader to found
                 value
16.              mark the temporary BACKBONE leader
17.              set the temporary BACKBONE leader to next leader
18.            else
19.              set the next leader of temporary BACKBONE leader
                 to LEAF with smallest node_id.
20.              mark this LEAF leader
21.         if I am a LEAF leader which can't find next leader
21.         find a child with smallest node_id from a previous BACKBONE
            leaders of my parent BACKBONE leader.
23.         if found set the next leader
24.         else set the next leader to starting BACKBONE leader
25.end
```

Fig. 4. Procedure executed by BACKBONE leaders and LEAF leaders which can't find next leader

```
1.Procedure ordinary_leaf_proc
2.begin
3.      set my state to LEAF
4.      Find a LEAF leader with same parent and nearest greater node_id.
5.      If found
6.         set my next leader to this LEAF leader's node_id and mark
           this LEAF.
7.end
```

Fig. 5. Procedure executed by LEAF leaders

clusterhead updates its next leader to next-next leader and continues its operation since it knows the global information of all clusterheads.

3.3 An Example Operation

Assume the MANET with clusterheads(leaders) in Fig. 6.a. Clusters are obtained using MCA. Nodes 65, 15, 98, 30, 40, 13, 28, 80, 74, 19, 51 and 99 are the leaders of clusters 1 to 12, respectively. Each clusterhead floods the *Leader_Info* message to the network. After each clusterhead receives the *Leader_Info* message of the others, minimum spaning tree in Fig. 6.a is constructed by all clusterheads. Nodes 65, 98, 40, 13, 80, 19 and 99 identify themselves as *LEAF* leaders since their degrees are all 1. Nodes 15, 30, 28, 74 and 51 identify themselves as *BACKBONE* leaders since their degrees are greater than 1. *BACKBONE* leaders are filled with black and *LEAF* leaders are filled with white as shown in Fig. 6.a.

To connect the *BACKBONE* nodes, a starting *BACKBONE* leader must be chosen. The criteria is to select the *BACKBONE* node which has the smallest connection to other *BACKBONE* leaders. Node 15 is connected to 30, 30 is connected to 15 and 28, 28 is connected to node 30 and node 74, node 74 is connected to node 28 and 51, 51 is connected to 74. Node 15 and 51 can be the choice for starting *BACKBONE* leader. 15 is selected because its *node_id* is smaller than 51. 15 selects the next leader as 30, 30 selects the next leader 28, operation continues in this way. The ending *BACKBONE* leader directs to its *LEAF* with the smallest *node_id*. These directions can be seen in Fig. 6.b with bold directed lines.

LEAF leaders of a *BACKBONE* leader are directed to each other from smallest to greatest. Node 19 is directed to 99, 13 is directed to 80, 65 is directed 98 as seen in Fig. 6.c with dotted directed lines.

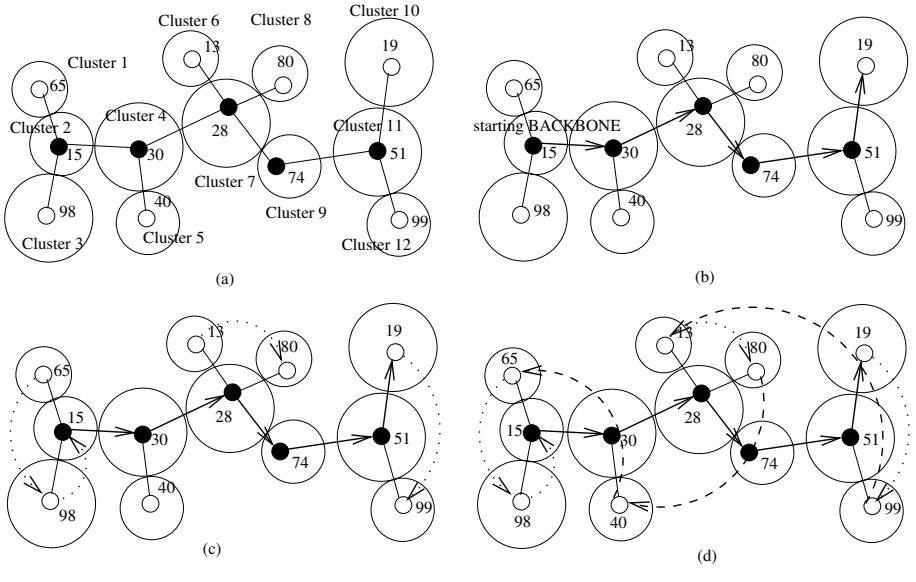

Fig. 6. An Example Operation

Lastly, *LEAF* leaders of different *BACKBONE* leaders are connected as in Fig. 6.d. Each *LEAF* leader which can not find the next leader, searches for a *LEAF* leader from the children of the previous *BACKBONE* leader of its parent *BACKBONE* leader. 99 is connected to 13, 80 is connected to 40, 40 is connected 65, 98 is connected to 15 shown with dashed lines in Fig. 6.d.

3.4 Analysis

Theorem 1. *Message complexity of the backbone formation algorithm is* $O(Kn)$.

Proof. Assume that we have n nodes in our network. K leaders flood the message to the network. Total number of messages in this case is Kn which means that message complexity has an upper bound of $O(n)$.

Theorem 2. *Time complexity of the backbone formation algorithm is* $O(Kn)$.

Proof. Assume that we have n nodes in our network. Flooding of K messages to the network takes Kn time.

4 Results

We implemented the distributed backbone formation algorithm with the *ns*2 simulator. Clustering is obtained using the MCA algorithm. Cluster size can be adjusted by the K heuristic of MCA. Position-based backbone formation algorithm is implemented.

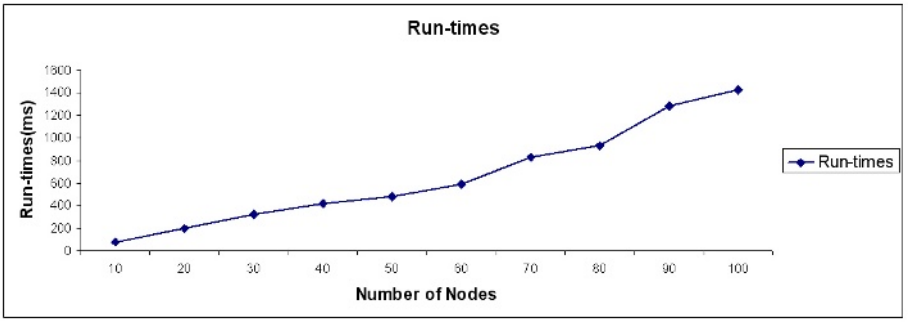

Fig. 7. Runtime Performance

Different size of flat surfaces are chosen for each simulation to create medium, small and very small distances between nodes. Medium, small and very small surfaces vary between 310m * 310m to 400m* 400m, 410m * 410m to 500m* 500m, 515m * 515m to 650m * 650m respectively. Random movements are generated for each simulation. Low, medium and high mobility scenarios are generated and node speeds are limited between 1.0m/s to 5.0m/s, 5.0m/s to 10.0m/s, 10.0m/s

to 20.0m/s respectively. K heuristic of merging clustering algorithm is changed to obtain different number of clusterheads. Round-trip delay as measured against the number of clusterheads, total number of nodes, mobility and surface area are recorded. As depicted in Fig. 7, the time complexity increases linearly and at worst, the backbone formation scheme is completed in 1.5s for a MANET with 100 nodes.

For a MANET with 50 nodes, number of clusterheads are selected from 3 to 8 to measure the round-trip delay in Fig. 8. A linear increase can be seen in Fig. 8 which starts from 35ms and ends in 65ms approximately.

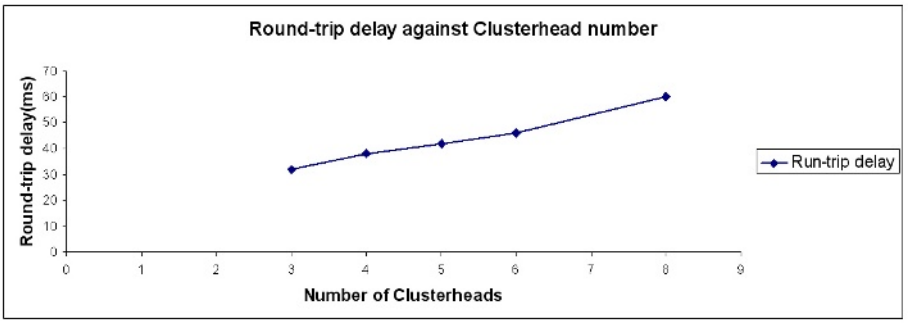

Fig. 8. Round-trip delay against number of clusterheads

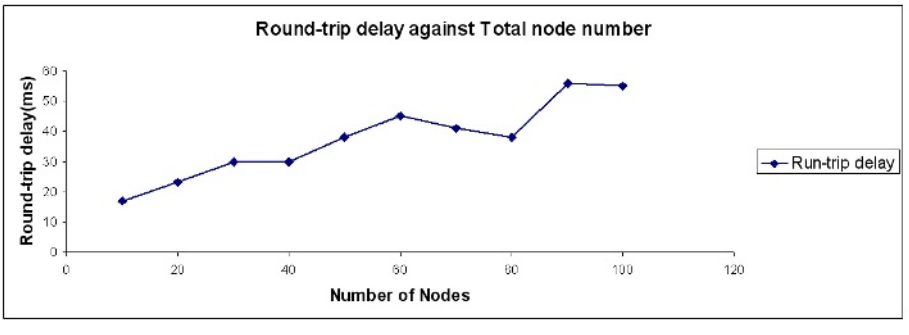

Fig. 9. Round-trip delay against number of nodes

Round-trip delay against total number of nodes is measured with constant 4 clusters and the total number of nodes are varied between 10 to 100 in Fig. 9. Round-trip delay times increase linearly from 20ms to 60ms approximately as shown in Fig. 9.

In small surface scenarios, the connectivity between nodes is higher because of small distances between the nodes and the connectivity between nodes causes a decrease in the routing delay. Fig. 10 shows the effects of distance between nodes to round-trip delay of the ring.

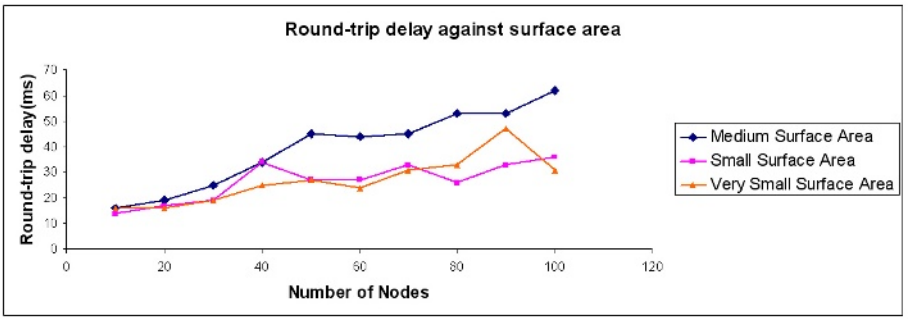

Fig. 10. Round-trip delay against surface area

Lastly, mobility parameter is changed to obtain the behavior of the algorithm with respect to mobility. Our algorithm results in approximate round-trip delay values for high mobile scenarios as shown in Fig. 11.

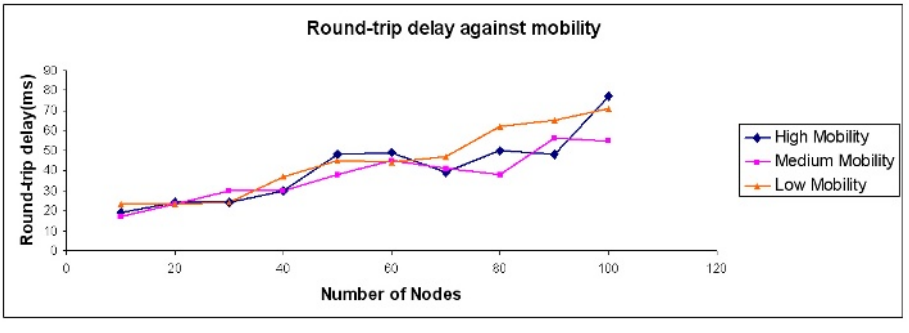

Fig. 11. Round-trip delay against Mobility

5 Conclusions

We proposed a new fully algorithm for backbone formation in MANETs and illustrated its operation. Our original idea is the construction of backbone architecture as a directed ring. The second contribution is to connect the clusterheads of a balanced clustering scheme which completes two essential needs of clustering by having balanced clusters and minimized routing delay. Besides these, the backbone formation algorithm is fault tolerant as the third contribution. The implementation results show that the algorithm is scalable in terms of its running time and round-trip delay against mobility, surface area, number of nodes and number of clusterheads. We are planning to experiment various *total order multicast* and *mutual exclusion* algorithms in such an environment where message ordering and mutual exclusion are provided by the clusterheads on behalf of the ordinary nodes of the MANET.

References

1. Yu, J.Y., Chong, P.H.J., "A survey of clustering schemes for mobile ad hoc networks", in Proc. IEEE Communications Surveys and Tutorials, 15531877, (2005).
2. Wu J., Li, H., L., "On Calculating Connected Dominating Set for Efficient Routing in Ad Hoc Wireless Networks", Proc. 3rd Intl. Wksp. Discrete Algorithms and Methods for Mobile Comp. and Commun., 714, (1999).
3. Chen Y.-Z. P., Liestman, A. L., "Approximating Minimum Size Weakly-Connected Dominating Sets for Clustering Mobile Ad Hoc Networks" , in Proc. 3rd ACM Intl. Symp. Mobile Ad Hoc Net. and Comp., 165-172, (2002).
4. Cokuslu, D., Erciyes, K. and Dagdeviren, O., "A Dominating Set Based Clustering Algorithm for Mobile Ad hoc Networks", ICCS 2006, Springer Verlag, LNCS, (2006).
5. Das, B., Bharghavan, V., "Routing in Ad Hoc Networks Using Minimum Connected Dominating Sets", in Proc. IEEE ICC97, 37680, 33(2), (1997).
6. Das B., Sivakumar, R. and Bharghavan, V., "Routing in Ad Hoc Networks Using a Spine", in Proc. IEEE Intl. Comp. and Commun. Net. 97, 120, (1997).
7. Lin, C. R. and Gerla, M. "Adaptive Clustering for Mobile Wireless Networks", IEEE JSAC, 1265-1275, 15, (1997).
8. Chiang, C.-C. et al., "Routing in Clustered Multihop, Mobile Wireless Networks with Fading Channel, in Proc. IEEE SICON97, (1997).
9. Yu, J. Y., Chong, P. H. J., "3hBAC (3-hop between Adjacent Clusterheads): a Novel Non-overlapping Clustering Algorithm for Mobile Ad Hoc Networks", in Proc. IEEE Pacrim03, 31821, 1, (2003).
10. Kwon, T. J. et al.,"Efficient Flooding with Passive Clustering an Overhead-Free Selective Forward Mechanism for Ad Hoc/Sensor Networks", in Proc. IEEE, 12101220, 91(8), Aug. 2003.
11. MaDonald, A. B., Znati, T. F., "A Mobility-based Frame Work for Adaptive Clustering in Wireless Ad Hoc Networks" , IEEE JSAC, 14661487, 17, Aug. (1999).
12. Basu, P., Khan, N. and Little, T. D. C., "A Mobility Based Metric for Clustering in Mobile Ad Hoc Networks", in Proc. IEEE ICDCSW 01, 41318, Apr. (2001).
13. McDonald, A. B., Znati, T. F., "Design and Performance of a Distributed Dynamic Clustering Algorithm for Ad-Hoc Networks", in Proc. 34th Annual Simulation Symp., 2735, (2001).
14. Amis, A. D., Prakash, R., "Load-Balancing Clusters in Wireless Ad Hoc Networks", in Proc. 3rd IEEE ASSET00, 2532, (2000).
15. Wu, J. et al., "On Calculating Power-Aware Connected Dominating Sets for Efficient Routing in Ad Hoc Wireless Networks", J. Commun. and Networks, 5970, 4(1), (2002).
16. Ryu, J.-H., Song, S., Cho, D.-H., "New Clustering Schemes for Energy Conservation in Two-Tiered Mobile Ad-Hoc Networks", in Proc. IEEE ICC01, 862866, 3, (2001).
17. Dagdeviren, O., Erciyes, K., Cokuslu, D., "A Merging Clustering Algorithm for Mobile Ad hoc Networks", ICCSA 2006, Springer Verlag LNCS, (2006).
18. Ohta, T., Inoue, S. and Kakuda, Y., "An Adaptive Multihop Clustering Scheme for Highly Mobile Ad Hoc Networks", in Proc. 6th ISADS03, (2003).
19. Chatterjee, M., Das, S. K. and Turgut, D., "An On-Demand Weighted Clustering Algorithm (WCA) for Ad hoc Networks", in Proc. IEEE Globecom00, 16971701, (2000).

20. Erciyes, K., "Cluster-based Distributed Mutual Exclusion Algorithms for Mobile Networks", EUROPAR 2004, Springer-Verlag, LNCS 3149, 933-940, (2004).
21. Gallagher, R. G., Humblet, P. A., AND Spira, P. M, "A Distributed Algorithm for Minimum-Weight Spanning Trees", ACM Transactions on Programming Languages and Systems 5, 66-77, (1983).