Merging Clustering Algorithms in Mobile Ad hoc Networks

Orhan Dagdeviren, Kayhan Erciyes and Deniz Cokuslu

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

Abstract. Clustering in mobile network is a widely used approach to ease implementation of various problems such as routing and resource management in mobile ad hoc networks (MANET)s. We first look at dominating set based clustering algorithms and minimum spanning tree (MST) based algorithms and then propose a new algorithm for clustering in MANETs. The algorithm we propose merges clusters to form higher level clusters by increasing their levels. We show the operation of the algorithm and analyze its time and message complexities.

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. Conserving energy is an important issue for MANETs as the nodes are powered by batteries only. 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 has many advantages in mobile networks. Clustering makes the routing process easier, also, by clustering the network one can build a virtual backbone which makes multicasting faster. However, the overhead of cluster formation and maintenance is not trivial. In a typical clustering scheme, the MANET is firstly partitioned into a number of clusters by a suitable distributed algorithm. A Cluster Head (CH) is then allocated for each cluster which will perform various task on behalf of the members of the cluster. The performance metrics of a clustering algorithm are the number of clusters and the count of the *neighbor nodes* which are the adjacent nodes between clusters that are formed [1].

In this study, we search various graph theoretic algorithms for clustering in MANETs and propose a new algorithm. One fundamental approach is the *Dominating Set based Clustering* which is reviewed in Section 2. Constructing *Minimum Spanning Trees* is another approach where part of a tree or a tree of a forest designates a cluster. Related work in this area is reviewed in Section 3, we illustrate our algorithm in Section 4 and the final section provides the conclusions drawn.

2 Background

2.1 Clustering Using Dominating Sets

A dominating set is a subset S of a graph G such that every vertex in G is either in S or adjacent to a vertex in S[2]. Dominating sets can be classified into three main classes, Connected Dominating Sets (CDS), Weakly Connected Dominating Sets (WCDS) and Independent Dominating Sets (IDS)[4].

- *Independent Dominating Sets:* IDS is a dominating set S of a graph G in which there are no adjacent vertices.
- *Weakly Connected Dominating Sets*(WCDS) : A weakly induced subgraph $(S)_w$ is a subset S of a graph G that contains the vertices of S, their neighbors and all edges of the original graph G with at least one endpoint in S. A subset S is a weakly-connected dominating set, if S is dominating and $(S)_w$ is connected [5].
- *Connected Dominating Sets:* A connected dominating set (CDS) is a subset S of a graph G such that S forms a dominating set and S is connected.

2.2 Dominating Set Algorithms

Various algorithms exist for clustering in IDS, WCDS and CDS.

Clustering Using IDS: By using independent dominating sets, one can guarantee that there are no adjacent cluster heads in the entire graph. This minimizes the number of dummy clusters in the network. Baker and Ephremides [6] proposed an independent dominating set algorithm called *highest vertex ID*. A very similar algorithm to the highest id algorithm is the *lowest id algorithm* by Gerla and Tsai [7]. Gerla and Tsai developed another algorithm to find the independent dominating sets called the *highest degree algorithm*.

Clustering Using WCDS: Although independent dominating sets are suitable for constructing optimum sized dominating sets, they have some deficiencies such as lack of direct communication between cluster heads. In order to obtain the connectivity between cluster heads, WCDSs can be used to construct clusters. The WCDS was first proposed for clustering in ad hoc networks by Chen and Liestman [8] called *zonal clustering*.

Clustering Using CDS: CDSs have many advantages in network applications such as ease of broadcasting and constructing virtual backbones [9], however, when we try to obtain a connected dominating set, we may have undesirable number of cluster heads. So in constructing connected dominating sets, our primary problem is minimum connected dominating set decision problem. Guha and Khuller [10] proposed two centralized greedy algorithms for finding suboptimal connected dominating sets. Das and Bharghavan [11] provided distributed implementations of Ghua and Khuller's algorithms [10]. Wu and Li [12], improved Das and Bhraghavan's distributed algorithm as a localized distributed algorithm for finding connected distributed sets in which each node only needs to know its distance-two neighbor [5]. Wu and Dai proposed an extended version of this algorithm which uses more general rules in order to cluster graphs [13].

3 Clustering Using a Minimum Spanning Tree

An undirected graph is defined as $G = (V, E)$, where V is a finite nonempty set and $E \subseteq V \times V$. The V is a set of nodes v and the E is a set of edges e. A graph G is connected if there is a path between any distinct e. A graph $G_S = (V_S, E_S)$ is a spanning subgraph of $G = (V, E)$ if $V_S = V$. A spanning tree of a graph is an undirected connected acyclic spanning subgraph. Intuitively, a spanning tree for a graph is a subgraph that has the minimum number of edges for maintaining connectivity [14].

3.1 Minimum Spanning Tree Algorithms

The idea is to group branches of a spanning tree into clusters of an approximate target size [15]. The resulting clusters can overlap and nodes in the same cluster may not be directly connected [5]. Gallagher, Humblet, Spira's Distributed Algorithm and Srivastava, Ghosh's k-tree core algorithm are two algorithms which construct distributed minimum spanning trees in MANETs.

Gallagher, Humblet and Spira's Distributed Algorithm: Gallagher, Humblet and Spira [16] proposed a distributed algorithm which determines a minimum-weight spanning tree for an undirected graph that has distinct finite weights for every edge. Aim of the algorithm is to combine small fragments into larger fragments with outgoing edges. A fragment of an MST is a subtree of the MST.

An outgoing edge is an edge of a fragment if there is a node connected to the edge in the fragment and one node connected that is not in the fragment. Combination rules of fragments are related with levels. A fragment with a single node has the level L = 0. Suppose two fragments F at level L and F' at level L';

- If L < L', then fragment F is immediately absorbed as part of fragment F. The expanded fragment is at level L'.
- Else if L = L' and fragments F and F' have the same minimum-weight outgoing edge, then the fragments combine immediately into a new fragment at level L+1
- Else fragment F waits until fragment F' reaches a high enough level for combination.

Under the above rules the combining edge is then called the core of the new fragment. The two essential properties of MSTs for the algorithm are:

- *Property* 1: Given a fragment of an MST, let e be a minimum weight outgoing edge of the fragment. Then joining e and its adjacent non-fragment node to the fragment yields another fragment of an MST.
- *Property* 2: If all the edges of a connected graph have different weights, then the MST is unique.

The algorithm defines three different states of operation for a node. The states are *Sleeping*, *Find* and *Found*. The states affect what of the following seven messages are sent and how to react to the messages. The messages are

Initiate, *Test*, *Reject*, *Accept*, *Report*(W), *Connect*(L) and *Change-core*. The identifier of a fragment is the core edge, that is, the edge that connected the two fragments together.A sample MANET and a minimum spanning tree constructed with Gallagher, Humblet, Spira's algorithm can be seen in Fig. 1 where any node other than the leaf nodes which are shown by black color depict a connected set of nodes. The upper bound for the number of messages exchanged during the execution of the algorithm is 5Nlog2 N +2E, where N is the number of nodes and E is the number of edges in the graph. A message contains at most one edge weight and $\log_2$ 8N bits. A worst case time for this algorithm is O(NlogN).

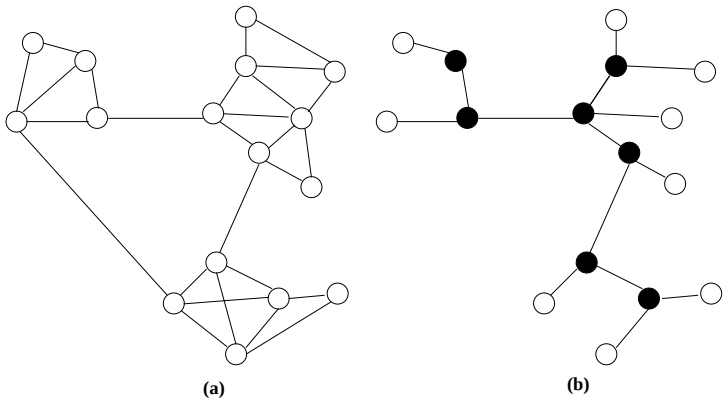

Fig. 1. (a) A MANET (b) Its Minimum Spanning Tree

4 Our Algorithm

We propose a distributed algorithm which finds clusters in a mobile ad hoc network. We assume that each node has distinct *node _id*. Moreover, each node knows its *cluster_leader_id*, *cluster_id* and *cluster_level*. *Cluster_id* is identified by the maximum *node_id* of the node in a cluster. *cluster_level* is identified by the number of the nodes in a cluster. *Cluster_leader_id* is identified by the *node_id* of the leader node in a cluster. *Cluster_ leader_id* is equal to the *cluster_id*. We assume that each node initially knows the cluster information of adjacent nodes. The local algorithm consists of sending messages over adjoining links, waiting for incoming messages and processing messages. The finite state machine of the algorithm is shown in Fig. 2.

The algorithm requires the sequence of messages as shown in Fig. 3. Firstly a node sends a *Poll_Node* message to a destination node. Destination node sends a *Node_Info* message back to originator node. Originator node then sends a *Connect_Ldr* or *Connect_Mbr* message to destination node to state it is the current leader or not. Destination node sends a *Ldr_ACK* or *Mbr_ACK* message to originator node.

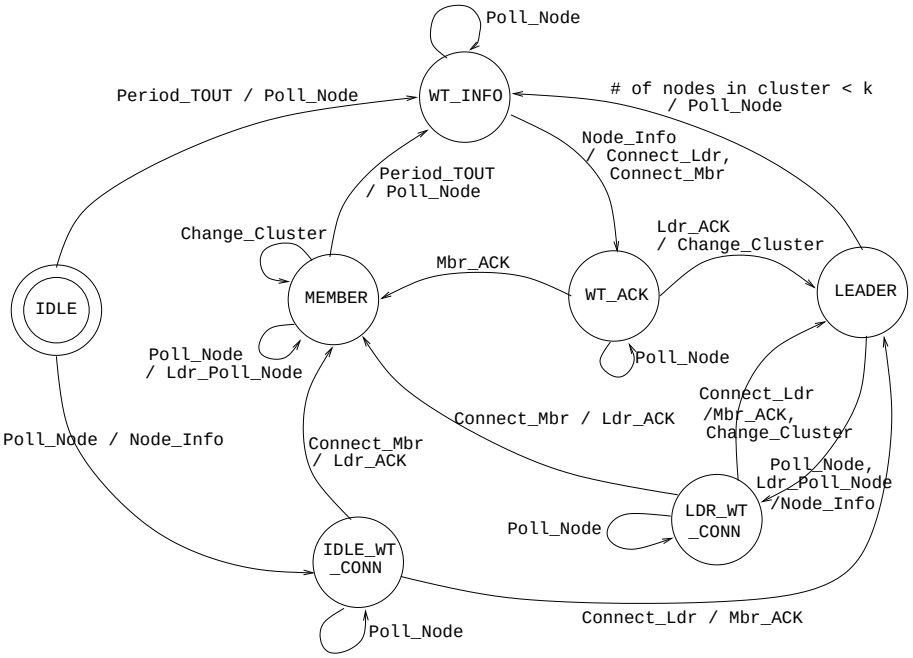

Fig. 2. Finite State Machine

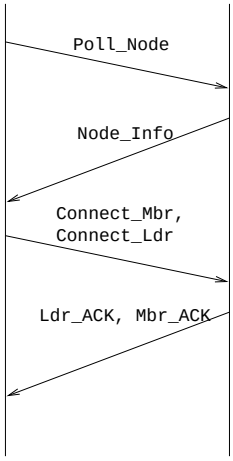

Fig. 3. Flow Diagram

Messages can be transmitted independently in both directions on an edge and arrive after an unpredictable but finite delay, without error and in sequence. Message types are *Poll_Node*, *Ldr_Poll_Node*, *Node_Info*, *Ldr_ACK*, *Mbr_ACK*, *Connect_Mbr*, *Connect_Ldr* and *Change_Cluster* as described below.

- *Poll_Node*: A cluster leader node will send *Poll_Node(node_id, cluster_level)* level) message to a destination node to begin the clustering operation. After sending *Poll_Node(node_id, cluster_level)* message, node waits for *Node_Info(node_id, cluster_level)* message from destination node.
- *Ldr_Poll_Node* : A cluster member node will send *Ldr_Poll_Node(node_id, cluster_level)* message to cluster leader node if cluster member node receives a *Poll_Node(node_id, cluster_level)* message from a node which is not in the same cluster.
- *Node_Info*: A cluster leader node will send *Node_Info(node_id, cluster_level)* message if it receives a *Poll_Node(node_id, cluster _level)* or *Ldr_Poll_Node(node_id, cluster_level)* message.
- *Connect_Mbr*: A cluster node will send *Connect_Mbr(node id)* message after it receives a *Node_Info(node_id, cluster_ level)* which has a smaller node_id than sender. Sender node of *Connect_Mbr(node_id)* message waits for *Ldr_ACK*.
- *Connect_Ldr*: A cluster node will send *Connect_Ldr(node id)* message after it receives a *Node_Info(node_id, cluster _level)* message which has a greater *node_id* than sender's *node_id*. Sender node of *Connect_Mbr(node_id)* message waits for *Mbr_ACK*.
- *Ldr_ACK*: A node will send *Ldr_ACK(node_id,cluster_level)* message when it receives a *Connect_Mbr* message. The last receiver node of the *Ldr_ACK* message is the leader of the cluster.
- *Mbr_ACK*: A node will send *Mbr_ACK* message when it receives a *Connect_Ldr* message. The receiver node of the *Mbr_ACK* message is a member of the cluster.
- *Change_Cluster*: A node will multicast a *Change_Cluster(node_id, cluster_level)* message after it receives a *Ldr_ACK* message. The leader of a cluster calculates new level and multicasts *Change_Cluster(node_id,cluster_level)* to all cluster member nodes to update their *cluster_id* and *cluster_level* information.
- *Period_TOUT*: This message can be regarded as an internal message. *Period_TOUT* occurs for every node in the network to start clustering operation periodically.

Every node in the network performs the same local algorithm. Each node can be either in *IDLE*, *WT_INFO*, *WT_ACK*, *MEMBER*, *LEADER*, *LDR_WT_CONN* or *IDLE_WT_CONN* states described below.

- *IDLE*: Initially all the nodes are in *IDLE* state. If *Period_TOUT* occurs, a node will send a *Poll_Node* message to destination node and will make a state transition to *WT_INFO* state. If *Poll_Node* message is received, the node will send a *Node_Info* message and make a state transition to *IDLE_WT_CONN*.

- *WT_INFO*: A node in *WT_INFO* state waits for *Node_Info* message. If a *Node_Info* message is received, the node will send a *Connect_Mbr* or *Connect_Ldr* message and make a state transition to *WT_ACK* state.
- *WT_ACK*: A node in *WT_ACK* state waits for a Mbr_ACK or *Ldr_ACK*. If *Mbr_ACK* is received, node will make a state transition to *MEMBER* state. If *Ldr_ACK* is received, node will multicast *CHANGE_LEADER* message and make a state transition to *LEADER* state.
- *MEMBER*: A cluster the member node is in the *MEMBER* state. If a *Poll_Node* message is received, the node will send *Ldr_Poll_Node* message to the leader node of the cluster. If a *Change_Cluster* message is received, the node will update its cluster information. If a *Period_TOUT* occurs, the node will send a *Poll_Node* message to destination node and will make a state transition to *WT_INFO* state.
- *LEADER*: When A cluster leader node is in the *LEADER* state, if a *Poll_Node* or a *Ldr_Poll_Node* is received, the node will send a *Node_Info* message and make a state transition to *LDR_WT_CONN* state. If the node finds that *number of nodes in cluster* $< k$, it will send a *Poll_Node* message and make a state transition to *WT_INFO*.
- *LDR_WT_CONN*: A node in *LDR_WT_CONN* state waits for *Connect_Mbr* or *Connect_Ldr* message. Previous state for this node is *LEADER*. If *Connect_Mbr* is received, node will make a state transition to *MEMBER* state. If *Connect_Ldr* is received, node will make a state transition to *LEADER* state.
- *IDLE_WT_CONN*: A node in *IDLE_WT_CONN* state waits for *Connect_Mbr* or *Connect_Ldr* message. Previous state for this node is *IDLE*. If *Connect_Mbr* is received, a node will make a state transition to *MEMBER* state. If *Connect_Ldr* is received, node will make a state transition to *LEADER* state.

Timeouts can occur when two nodes are communicating. If a timeout occurs at a node either in *IDLE*, *WT_INFO*, *WT_ACK* or *IDLE_WT_CONN* states, it returns back to *IDLE* state, a node in *LDR_WT_CONN* state returns back to *LEADER* state, a node either in *LEADER* or *MEMBER* states doesn't change its state.

4.1 Procedures

Procedures executed when a node receives a message is given in this section. Procedures in Fig. 4, Fig. 5, Fig. 6 and Fig. 7 aren't standard procedures which need no extra operation other than state transition. When a node in *IDLE_WT_CONN* state receives a *Connect_Ldr*, it will calculate the *new_cluster_level* and multicast *Change_Cluster* message to all cluster elements. This situation is shown in Fig. 4. When a node in *MEMBER* state receives a *Change_Cluster* message, it will update its all information as shown in Fig. 5. When a node in *WT_ACK* state receives a *Leader_ACK*, it will calculate the *new_cluster_level* and multicast *Change_Cluster* message to all cluster elements. This situation is shown in Fig. 6 and Fig. 7

```
1.Procedure Connect_Ldr_IDLE_WT_CONN(node_id, cluster_level)
2.begin
3.  send Mbr_ACK to node_id
4.  new_cluster_level=my cluster_level+ cluster_level
5.  multicast a Change_Cluster(my_node_id,new_cluster_level)
    message to all cluster members, which are in the cluster with
    cluster_id is equal to node id and cluster_id is equal to my
    node id
6.  set my state to LEADER
7.end
```

Fig. 4. Procedure executed when a *Connect_Ldr* is received from a node in *IDLE_WT_CONN* state.

```
1.Procedure Change_Cluster_MEMBER(node_id, cluster_level)
2.begin
3.  my_cluster_id=node_id
4.  my_cluster_leader_id=node_id
5.  my_cluster_level=cluster_level
7.end
```

Fig. 5. Procedure executed when a *Change_Cluster* is received from a node in *MEMBER* state.

```
1.Procedure Ldr_ACK_WT_ACK(node_id,cluster_level)
2.begin
3.  new_cluster_level=my_cluster_level+cluster_level
4.  multicast a Change_Cluster(my_node_id,new_cluster_level)
    message to all cluster members, which are in the cluster with
    cluster_id is equal to node_id and cluster_id is equal to my
    node_id
5.  set my state to LEADER
6.end
```

Fig. 6. Procedure executed when a *Ldr_ACK* is received from a node in *WT_ACK* state.

```
1.Procedure Connect_Ldr_LDR_WT_CONN(node_id,cluster_level)
2.begin
3.  new_cluster_level=my_cluster_level+cluster_level
4.  send Mbr_ACK message to node_id
5.  multicast a Change_Cluster(my_node_id,new_cluster_level)
    message to all cluster members, which are in the cluster with
    cluster_id is equal to node_id and cluster_id is equal to my_node_id
6.  set my state to LEADER
7.end
```

Fig. 7. Procedure executed when a *Ldr_Poll_Node* is received from a node in *LEADER* state.

4.2 An Example Operation

Assume the mobile network in Fig. 8. Initially all the clusters are in *IDLE* state. *Period_TOUT* occurs in Node 1, Node 3, Node 4 , Node 9 and Node 12. Node 1 sends a *Poll_Node* message to Node 7 and sets its state to *WT_INFO*. Node 7 receives the *Poll_Node* message and sends *Node_Info* message to Node 1. Node 7 sets its state to *IDLE_WT_CONN*. Node 1 receives the *Node_Info* message and sends a *Connect_Ldr* message to Node 7 since the *node_id* of Node 7 is greater than node 1. Node 1 sets its state to *WT_ACK*. Node 7 receives the *Connect_Ldr* message and sends a *Mbr_ACK* message to Node 1. Node 1 receives the message and sets its state to *MEMBER*. Node 7 sends *Change_Cluster* message to Node 1 indicating that new cluster is formed between and Node 1 and Node 7. Node 8 and Node 9, Node 2 and Node 4 , Node 11 and Node 5, Node 3 and Node 6 are connected same as Node 1 and Node 2 to form clusters with level 2.

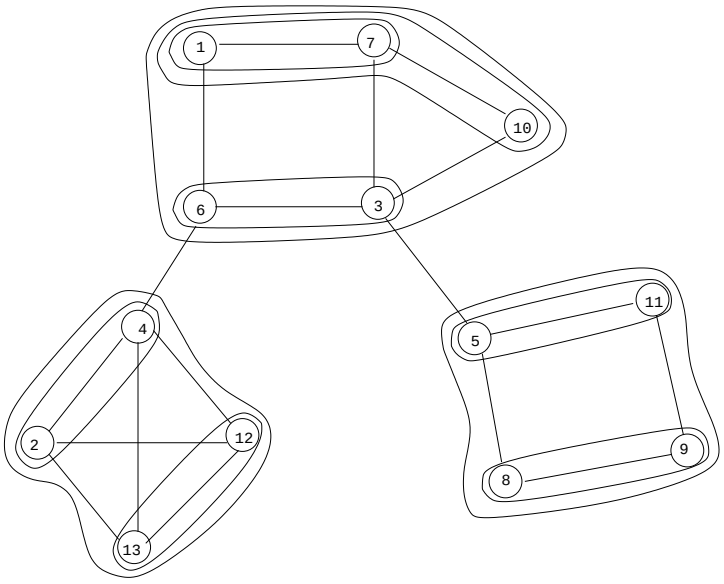

Fig. 8. Clusters obtained using our algorithm

After clusters with level 2 are formed, Node 10 in *IDLE* state sends a *Poll_Node* message to Node 7. Node 10 sets its state to *WT_INFO*. Node 7 in *LEADER* state receives *Poll_Node* message and sends a *Node_Info* message to Node 10. Node 7 sets its state to *LDR_WT_CONN*. Node 10 in *WT_INFO_STATE* receives *NODE_INFO* message from Node 7 and sends a *Connect_Mbr* message to Node 7. Node 10 sets its state to *WT_ACK*. Node 7 receives *Connect_Mbr* and sends *Ldr_ACK* message to Node 10. Node 7 sets its state to *MEMBER*. Node 10 in *WT_ACK* state receives *Ldr_ACK* message and multicasts *Change_Cluster* message to Node 1 and Node 7 to update new cluster information. Node 10 sets

its state to *LEADER*. At the same time Node 13 in *LEADER* state sends a *Poll_Node* message to Node 4. 12, 13 and 2, 4 forms a new cluster as shown before. Beside this 5, 11 and 8, 9 are connected to form new clusters.

Table 1. Cluster Formation

Iteration	*A*	*B*	*C*
1	1 7 10 6 3	2 13	5 9
2	1-7 10 6-3	2-4 13-12	5-11 9-8
3	1-7-10 6-3	2-4-13-12	5-11-9-8
4	1-7-10-6-3	*No Change*	*No Change*

Node 6 in *LEADER* state sends a *Poll_Node* message to Node 1. Node 6 changes its state to *WT_INFO*. Node 1 in *MEMBER* state receives the *Poll_Node* message and sends a *Ldr_Poll_Node* message to Node 10. Node 10 in *LEADER* state receives the *Ldr_Poll_Node* message and sends a *Node_Info* message to Node 6. Node 10 sets its state to *LDR_WT_CONN* state. Node 6 in *WT_INFO* state receives the *NODE_INFO* and sends a *Connect_Ldr* message. Node 6 sets its state to *WT_ACK*. The cluster formation scheme is continued as shown in finite state machine in Fig. 2. Lastly the clusters in Fig. 8 are summarized in Tab. 1.

4.3 Analysis

Theorem 1. *Time complexity of the clustering algorithm has a lower bound of* $\Omega(logn)$ *and upperbound of* $O(n)$.

Proof. Assume that we have n nodes in the mobile network. Best case occurs when each node can merge with each other exactly. To double member count at each iteration such that Level 1 clusters are connected to form Level 2 clusters. Level 2 Clusters are connected to form Level 4 Clusters and so on. The clustering operation continues until the to Cluster Level becomes m.The lower bound is $\Omega(logN)$. Worst case occurs when a cluster is connected to a Level 1 cluster at each iteration. Level 1 cluster is connected to a Level 1 cluster to form a Level 2 cluster, Level 2 cluster is connected to a Level 1 cluster to form a Level 3 cluster and so on. The clustering operation continues until the Cluster Level becomes n. The upper bound is therefore $O(n)$.

Theorem 2. *Message complexity of our algorithm is O(n).*

Proof. Assume that we have n nodes in our network. For every merge operations of two clusters 4 messages (*Poll_Node*, *Node_Info*, *Connect_Ldr/Connect_Mbr*, *Leader_ACK/Member_ACK*) are required. Total number of messages in this case is 4n which means message complexity has an upper bound of O(n).

5 Conclusions

We showed that dominating set based clustering is a fundamental approach whereas MST based clustering is also a promising approach. We also proposed a new algorithm for clustering in MANETs and illustrated its operation. We showed the implementation of the algorithm and analyzed its time and message complexity. We are in the process of implementing the algorithm proposed in a simulated environment. We are also currently working on using a combination of heuristics to partition a MANET and we are planning to experiment various *total order multicast* algorithms in such an environment where message ordering is provided by the cluster heads on behalf of the ordinary nodes of the MANET.

References

1. Nocetti, F., B., Gonzalez, J., S. and Stojmneovic, I., Connectivity Based k-Hop Clustering in Wireless Networks, Telecommunication Systems, (22)1-4,(2003), 205-220.
2. D. West, Introduction to Graph Theory, Second edition, Prentice Hall, Upper Saddle River, N.J., 2001.
3. Y.-Z. P. Chen and A. L. Liestman, 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., June 2002, pp. 16572.
4. T. W. Haynes, S. T. Hedetniemi, and P. J. Slater, Domination in graphs, Advanced Topics, Marcel Dekker, Inc., 1998.
5. Yuanzhu Peter Chen, Arthur L. Liestman, Jiangchuan Liu; Clustering Algorithms for Ad Hoc Wireless Networks
6. Baker, D.; Ephremides, A.; The Architectural Organization of a Mobile Radio Network via a Distributed Algorithm, Communications, IEEE Transactions on [legacy, pre - 1988] Volume 29, Issue 11, Nov 1981 Page(s):1694 - 1701
7. Mario Gerla, Jack Tzu-Chieh Tsai: Multicluster, mobile, multimedia radio network. Wireless Networks 1(3): 255-265 (1995)
8. Yuanzhu Peter Chen, Arthur L. Liestman; A Zonal Algorithm for Clustering Ad Hoc Networks
9. I. Stojmenovic, M. Seddigh, and J. Zunic, Dominating sets and neighbor elimination-based broadcasting algorithms in wireless networks, IEEE Transactions on Parallel and Distributed Systems, 13 (2002), pp. 14-25.
10. S. Guha and S. Khuller; Approximation Algorithms for Connected Dominating Sets, Springer-Verlag New York, LLC, ISSN: 0178-4617 (Paper) April 1998
11. Das, B.; Bharghavan, V.; Routing in ad-hoc networks using minimum connected dominating sets, Communications, 1997. ICC 97 Montreal, 'Towards the Knowledge Millennium'. 1997 IEEE International Conference on Volume 1, 8-12 June 1997 Page(s):376 - 380 vol.1
12. Jie, Wu, Hailan, Li; A Dominating-Set-Based Routing Scheme in Ad Hoc Wireless Networks, Springer Science+Business Media B.V., Formerly Kluwer Academic Publishers B.V. ISSN: 1018-4864 (Paper) September 2001
13. Jie, Wu, Fei, Dai; An extended localized algorithm for connected dominating set formation in ad hoc wireless networks. IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 15, NO. 10, OCTOBER 2004

14. Grimaldi, R. P. "Discrete and Combinatorial Mathematics, An Applied Introduction", Addison Wesley Longman, Inc., 1999.
15. S. Banerjee and S. Khuller, A clustering scheme for hierarchical routing in wireless networks, Tech. Report CS-TR-4103, University of Maryland, College Park, February 2000.
16. 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, 1 (Jan. 1983), 6677.