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 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 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, the count of the nodes in a cluster and the count of the *neighbor nodes* which are the adjacent nodes between the formed clusters [1].

In this study, we search various graph theoretic algorithms for clustering in MANETs and propose a new algorithm. Constructing *Minimum Spanning Trees* is an important approach where part of a tree or a tree of a forest designates a cluster. Related work in this area is reviewed in Section 2, we describe and illustrate the operation of our algorithm in Section 3 and the final section provides the conclusions drawn.

G. Chakraborty (Ed.): ICDCIT 2005, LNCS 3816, pp. 56–61, 2005.
© Springer-Verlag Berlin Heidelberg 2005

2 Background: 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 [3]. The idea is to group branches of a spanning tree into clusters of an approximate target size [4]. The resulting clusters can overlap and nodes in the same cluster may not be directly connected [2]. Gallagher, Humblet, Spira's Distributed Algorithm [5] and Srivastava, Ghosh's k-tree core Algorithm [6] are two algorithms which construct distributed minimum spanning trees in MANETs.

Gallagher, Humblet and Spira's Distributed Algorithm: Gallagher, Humblet and Spira [5] 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 upper bound for the number of messages exchanged during the execution of the algorithm is $5N\log_2 N + 2E$, 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) [5].

3 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. 1.

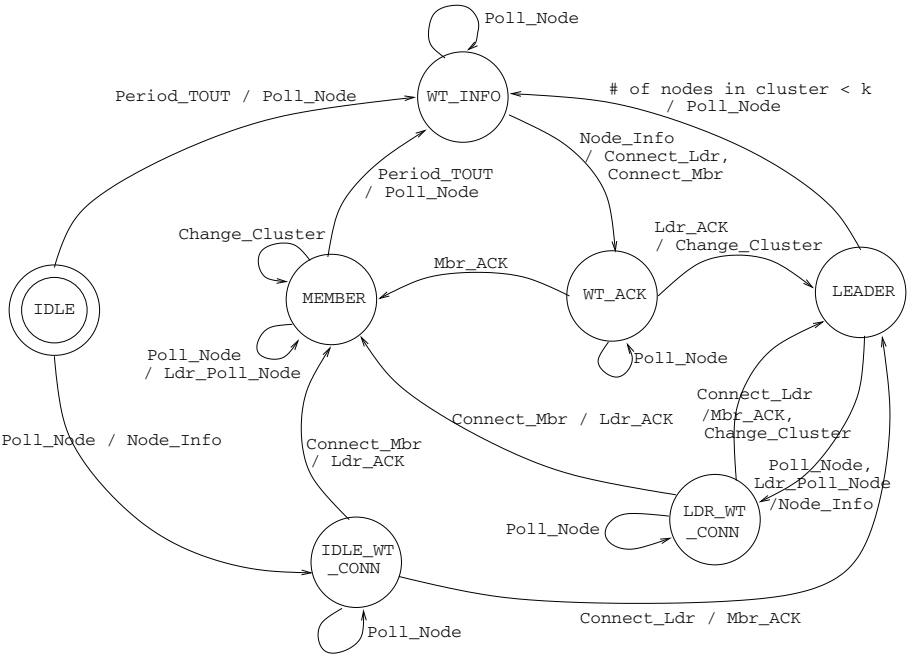

Fig. 1. Finite State Machine of the Algorithm

The algorithm requires the following sequence of messages. 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. *Ldr_ACK* message shows that the originator node will become the new leader. *Mbr_ACK* message shows that the originator node will become the member of the new cluster.

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.

A cluster member node will send *Ldr_Poll_Node* message to the cluster leader node if the cluster member node receives a *Poll_Node* message from a node which is not in the same cluster. A node will multicast a *Change_Cluster* to all cluster member nodes to update their *cluster_id* and *cluster_level*. *Period_TOUT* 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.

Initially all the nodes are in *IDLE* state before *Period_TOUT* occurs. A node in *WT_INFO* state waits for *Node_Info* message. A node in *WT_ACK* state waits for a Mbr_ACK or *Ldr_ACK*. A node in *LDR_WT_CONN* state waits for *Connect_Mbr* or *Connect_Ldr* message. A node in *IDLE_WT_CONN* state waits for *Connect_Mbr* or *Connect_Ldr* message. After the clustering operation is completed the nodes are either in *MEMBER* or *LEADER* state.

Timeouts can occur during communication. 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.

3.1 An Example Operation

Assume the mobile network in Fig. 2. 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.

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* mes-

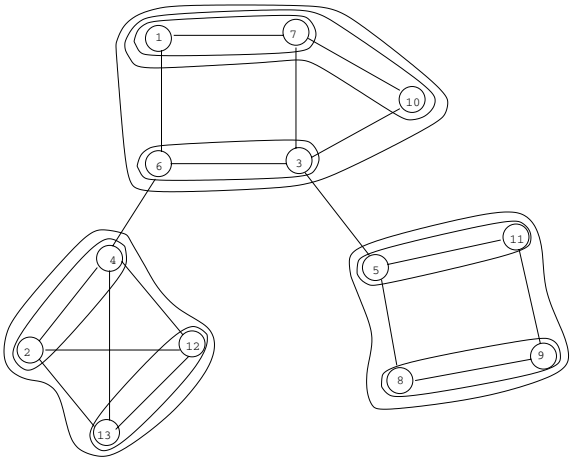

Fig. 2. Clusters obtained using our algorithm

sage 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. 1. Lastly the clusters in Fig. 2 are summarized in Tab. 1.

3.2 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 5 messages (*Poll_Node*, *Node_Info*, *Connect_Ldr/Connect_Mbr*, *Leader_ACK/Member_ACK*, *Change_Cluster*) are required. Total number of messages in this case is $5n$ which means message complexity has an upper bound of $O(n)$.

4 Conclusions

We 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. Our algorithm has a similar but more simplified structure than Gallagher's Algorithm [5]. The algorithm has a lower complexity and also we aim at forming clusters whereas the latter tries to find an MST. We are in the process of implementing the algorithm proposed in a simulated environment. We are planning to experiment various total order multicast and distributed mutual exclusion algorithms in such an environment where message ordering and synchronization are provided by the cluster heads on behalf of the ordinary nodes of the MANET.

References

1. Nocetti, F., B. et al, Connectivity based k-Hop clustering in wireless networks, Telecommunication Systems, (22)1-4,(2003), 205-220.
2. Chen , Y. P., Liestman, A. L., Liu, J., Clustering algorithms for ad hoc wireless networks, in Ad Hoc and Sensor Networks ed. Pan, Y. ,Xiao, Y., Nova Science Publishers, 2004.
3. Grimaldi, R. P., Discrete and Combinatorial Mathematics, An Applied Introduction, Addison Wesley Longman, Inc., 1999.
4. Banerjee, S., Khuller, S., A clustering scheme for hierarchical routing in wireless networks, CS-TR-4103, Univ. of Maryland, College Park, February 2000.
5. Gallagher, R. G., Humblet, P. A., Spira, P. M., A distributed algorithm for minimum-weight spanning trees, ACM Trans. on Programming Languages and Systems, (5)1, (1983), 66-77.
6. Srivastava, S., Ghosh, R., A cluster based routing using a k-tree core backbone for mobile ad hoc networks, Proceedings DIALM, (2002), 14-23.