A Merging Clustering Algorithm for 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 propose a new fully distributed algorithm for clustering in MANETs that 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 and provide results in the simulation environment of *ns*2. Our results conform that the algorithm proposed is scalable and has a lower time and message complexities than the other algorithms.

1 Introduction

MANETs consist of dynamic collection of nodes with rapidly changing topologies of wireless links. These networks have many important applications including disaster recovery operations, military operations and personal area networking. An important way to support efficient communication between nodes of a MANET is to develop a wireless mobile backbone architecture. Nodes in a MANET are powered by batteries only. Therefore, amount of communication should be minimized to avoid a premature drop out of a node from the network. Clustering has become an important approach to manage MANETs. The clustering problem can be described as classifying nodes in a MANET hierarchically into equivalence classes with respect to certain attributes such as geographical regions or small neighborhood of 1 or 2 hops from special nodes called the clusterheads[1]. Clusterheads may perform routing, typically by forming a virtual backbone with other clusterheads, network management and resource allocation for their cluster members by cooperating with other clusterheads. 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 [2].

In this study, we propose an algorithm for clustering in MANETs using merging as in constructing *Minimum Spanning Tree*s 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 illustrate our algorithm in Section 3, provide implementation results in Section 4 and the final section provides the conclusions drawn.

M. Gavrilova et al. (Eds.): ICCSA 2006, LNCS 3981, pp. 681–690, 2006.

2 Background

2.1 Clustering Using a Minimum Spanning Tree

An undirected graph is defined as $G = (V, E)$, where V is a finite nonempty set and $E \subseteq V \times V$. 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 v. 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 minimum spanning tree(MST) for a graph is a subgraph that has the minimum number of edges for maintaining connectivity [3].

Spanning Tree Algorithms. 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 [5]. Gallagher, Humblet and Spira [6], Awerbuch [7], Yao-Nan Lien [8], Ahuja and Zhu [9], Garay, Kutten and Peleg [10], Banerjee and Khuller [4] have all proposed distributed spanning tree based algorithms and Srivastava and Ghosh's [11] distributed k-tree core algorithm also constructs a distributed spanning tree.

Gallagher, Humblet and Spira's Distributed Algorithm: Gallagher, Humblet and Spira [6] 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

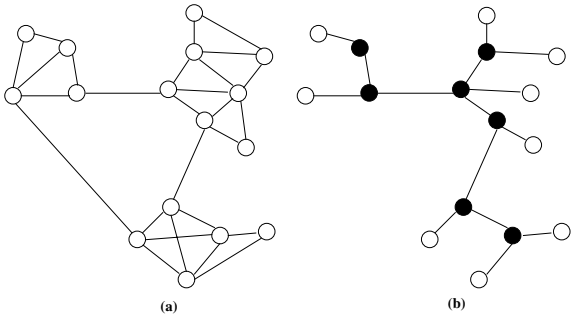

Fig. 1. (a) A MANET (b) Its Minimum Spanning Tree

messages are sent and how to react to the messages: *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 connects the two fragments together. A sample MANET and a minimum spanning tree constructed with Gallagher, Humblet, Spira's algorithm [6] 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 $5Nlog_2N + 2E$, 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 emphlog$_2$8N bits. A worst case time for this algorithm is $O(NlogN)$.

3 Our Algorithm

3.1 General Idea of the Algorithm

The distributed algorithm proposed finds clusters in a MANET by merging the clusters to form higher level clusters as mentioned in Gallagher, Humblet, Spira's algorithm [6]. However, we focus on the clustering operation by discarding minimum spanning tree. This reduces the message complexity as explained in Section 3.4 . The second contribution is to use upper and lower bound heuristics for clustering operation which results in balanced number of nodes in the clusters formed.

3.2 Description of the Algorithm

We assume that each node has distinct *node_id*. Moreover, each node knows its *cluster_leader_id*, *cluster_id* and *cluster_level*. *Cluster_level* is identified by the number of the nodes in a cluster. Leader node is the node with maximum *cluster_id*. *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*. 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.

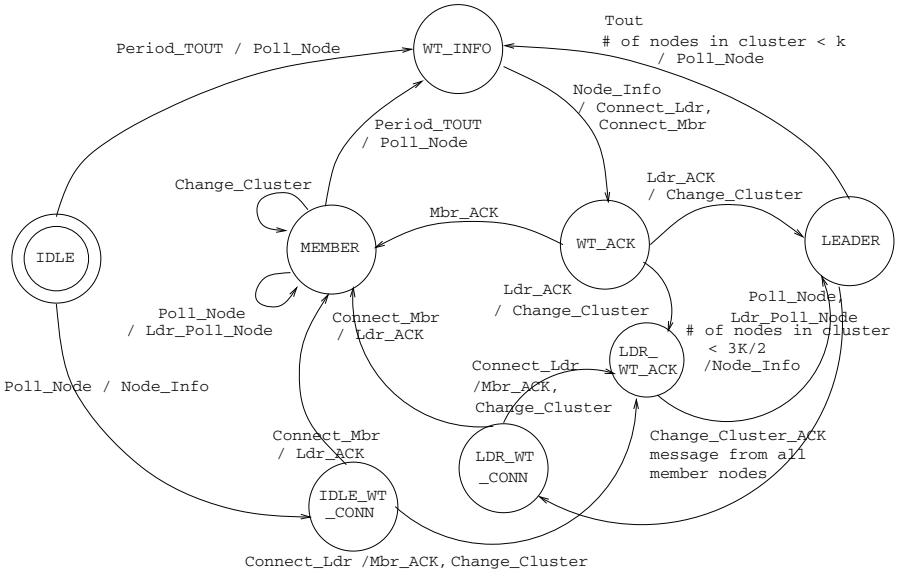

Fig. 2. Finite State Machine of the Merging Clustering Algorithm

The algorithm requires the sequence of messages as 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. We assume that the underlying network provides broadcast communication. After the above message exchange, the new leader node multicasts a *Change_Cluster* message to all cluster nodes and waits for *Change_Cluster_ACK* message from all cluster nodes.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*, *Change_Cluster* and *Change_Cluster_ACK* as described below.

- *Poll_Node*: A cluster leader node will send *Poll_Node (node_id, cluster_level)* message to a destination node to begin the clustering operation.
- *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.

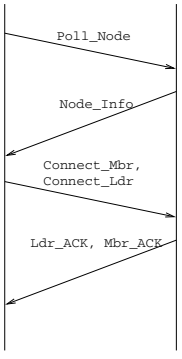

Fig. 3. Message Flow Diagram

- *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.
- *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*.
- *Ldr_ACK*: A node will send *Ldr_ACK (node_id, cluster_level)* message when it receives a *Connect_Mbr* message.
- *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.
- *Change_Cluster_ACK*: A node will send a *Change_Cluster_ACK* message after it receives *Change_Cluster* message.
- *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 nodes are in *IDLE* state. If *Period_TOUT* occurs, node sends a *Poll_Node* message to destination node and will make a state transition to *WT_INFO* state.
- *WT_INFO*: A node in *WT_INFO* state waits for *Node_Info* message.

- *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.
- *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 firstly check the $3K/2$ parameter to decide on the clustering operation. If cluster level is smaller, node will send a *Node_Info* message and make a state transition to *LDR_WT_CONN* state.
- *LDR_WT_CONN*: A node in *LDR_WT_CONN* state waits for *Connect_Mbr* or *Connect_Ldr* message. 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. If *Connect_Mbr* is received, the node will make a state transition to *MEMBER* state.
- *LDR_WT_ACK*: A node in *LDR_WT_ACK* state waits for *Change_Cluster_ACK* messages from all member nodes in the new cluster.

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

3.3 An Example Operation

Assume the mobile network in Fig. 4. K parameter is given as 4. 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 1 sends a *Change_Cluster_ACK* message to Node 7 which shows that the clustering operation between Node 1 and Node 7 is completed.Node 8 and Node 9, Node 2 and Node 4 , Node 11 and Node 5,

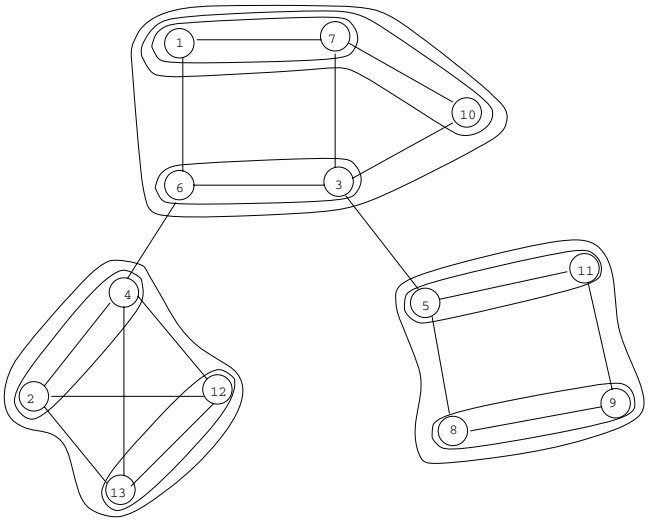

Fig. 4. Clusters obtained using the clustering algorithm

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 checks the $3K/2$ parameter. Since cluster level of Node 7 is smaller than K, Node 7 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 *LDR_WT_ACK*. Node 1 and Node 7 receives *Change_Cluster* messages and replies with *Change_Cluster_ACK* messages. Node 10 receives *Change_Cluster_ACK* messages and 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. The cluster formation scheme is continued as shown in finite state machine in Fig. 2. The formation of clusters in Fig. 4 are depicted in Tab. 1.

3.4 Analysis

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

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*

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 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 the clustering 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. K *Change_Cluster* messages and K *Change_Cluster_ACK* messages are also required. Total number of messages in this case is $(4+2K)n/K$ which means that message complexity has an upper bound of $O(n)$.

Theorem 3. *Cluster Levels vary between* K *and* $5K/2 - 2$.

Proof. A cluster leader periodically polls its neighbors until it reaches the cluster level with K. This guarantees the minimum cluster level with K.

Assume the scenario that a cluster leader with a cluster level with *K-1* tries to connect to another cluster with level *3K/2-1*. Consequently a new cluster with level *5K/2-2* will be formed.

4 Results

We implemented the merging clustering algorithm with *ns*2 simulator. A flat surface of 650m*650m is chosen for the simulation. Dynamic Source Distance Vector Routing is used as the routing protocol.

Random movements are generated for each simulation. Node speeds are limited between 1.0m/s and 5.0m/s. The computational run times, cluster node counts(cluster levels) and total edge cuts are recorded. Fig. 5 displays the runtime results of the merging clustering algorithm ranging from 10 to 100 nodes.

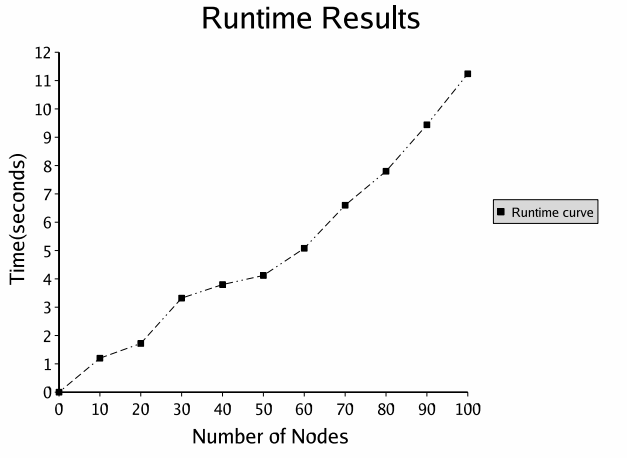

Fig. 5. Runtime Performance

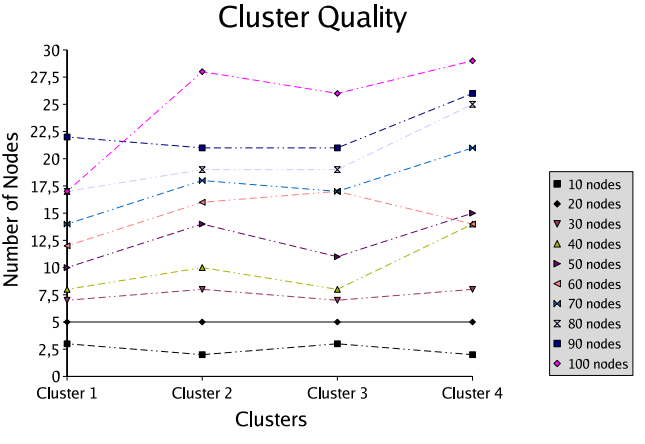

Fig. 6. Cluster Node Counts

Fig. 6 depicts the number of nodes in each cluster formed by the merging clustering algorithm. As depicted in Fig. 5, the time complexity increases linearly as also shown in Theorem 1. Clusters have similar number of nodes showing a balanced partitioning in Fig. 6.

5 Conclusions

We proposed a new fully distributed algorithm for clustering in MANETs and illustrated its operation. Our original idea is to focus on the clustering operation by discarding the details of minimum spanning tree algorithms to reduce time and message complexity. The second contribution is the usage of lower and

upper bound heuristics which results in balanced number of nodes in the clusters formed. The implementation results obtained conform with the theoretical analysis and show that the algorithm is scalable in terms of its running time and produces evenly distributed clusters. We are planning to experiment various *total order multicast* and *mutual exclusion* 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. Krishna, P., Vaidya, N. H., Chatterjee, M., Pradhan, D. K. : A Cluster-based Approach for Routing in Dynamic Networks, in SIGCOMM Computer Communications Review (CCR), (1997).
2. Nocetti, F., B., Gonzalez, J. S., Stojmneovic, I. : Connectivity Based k-Hop Clustering in Wireless Networks, Telecommunication Systems, (2003), (22)1-4, 205-220.
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, Tech. Report CS-TR-4103, University of Maryland, College Park, (2000).
5. Chen, Y. P., Liestman, A. L., Liu, J. : Clustering Algorithms for Ad Hoc Wireless Networks, in Ad Hoc and Sensor Networks ed. Y. Pan and Y. Xiao, Nova Science Publishers, (2004).
6. Gallagher, R. G., Humblet, P. A., Spira, P. M. : A Distributed Algorithm for Minimum-Weight Spanning Trees, ACM Transactions on Programming Languages and Systems 5, (1983), 66-77.
7. Awerbuch, B. : Optimal Distributed Algorithms for Minimum Weight Spanning Tree, Counting, Leader Election and related problems. , Proc. of the 9th Annual ACM Symposium on Theory of Computing, (1987), 230-240.
8. Lien, Y. N. : A New Node-Join-Tree Distributed Algorithm for Minimum Weight Spanning Trees , Proc. of the 8th International Conference on Distributed Computing Systems, (1988), 334-340.
9. Ahuja, M., Zhu, Y. : A Distributed Algorithm for Minimum Weight Spanning Trees Based on Echo Algorithms, Proc. of the 9th International Conference on Distributed Computing Systems, (1989).
10. Garay, J.A., Kutten, S., Peleg, D. : A sub-linear time distributed algorithm for minimum-weight spanning trees, Proc. of the 34th Annual Symposium on Foundations of Computer Science, (1993), 659-668,
11. Srivastava, S., Ghosh, R. K. : Distributed Algorithms for finding and maintaining a k-tree core in a dynamic network, Information Processing Letters, (2003), 88(4), 187-194.