
Graph Matching based Distributed Clustering and Backbone Formation Algorithms for Sensor Networks

ORHAN DAGDEVIREN¹, KAYHAN ERCIYES²

¹*Izmir Institute of Technology,
Computer Eng. Dept., Urla, Izmir 35430, Turkey
Tel: +90 232 750 78 67, Fax: +90 232 750 78 62*

²*Izmir University
Computer Eng. Dept., Gursel Aksel Bulv., No. 14, Uckuyular, Izmir 35350, Turkey
Tel: +90 232 246 49 49, Fax: +90 232 224 09 09
Email: orhandagdeviren@iyte.edu.tr*

Clustering is a widely used technique to manage the essential operations such as routing and data aggregation in wireless sensor networks (WSN)s. We propose two new graph theoretic distributed clustering algorithms for WSNs that use weighted matching method for selecting strong links. To the best of our knowledge, our algorithms are the first attempts that use graph matching for clustering. The first algorithm is divided into rounds, extended weighted matching operation is executed by nodes in each round, thus the clusters are constructed synchronously. The second algorithm is the enhanced version of the first algorithm, which provides not only clustering but also backbone formation in an energy efficient and asynchronous manner. We show the operation of the algorithms, analyze them, provide the simulation results in ns2 environment. We compare our proposed algorithms with the other graph theoretic clustering algorithms and show that our algorithms select strong communication links and create controllable number of balanced clusters while providing low energy consumptions. We also discuss possible applications that may use the structure provided by these algorithms and the extensions to the algorithms.

Received 00 Month 2009; revised 00 Month 2009

1. INTRODUCTION

WSNs do not have any fixed infrastructure and consist of sensor nodes that perform sensing and communicating tasks. Habitat monitoring, military surveillance and target tracking are example application types in WSNs. Clustering the network to construct a robust communication structure is a significant research area in WSNs. Clustering and backbones using clusters are provided in WSNs in order to decrease the number of messages and total time spent for routing the sensed data to the sink. By clustering the network, an efficient topology is constructed which makes routing and data aggregation tasks easier. In clustering schemes, each node is classified as either cluster head or cluster member. Cluster members are ordinary nodes whereas cluster heads perform various task on behalf of the members of the clusters.

A WSN can be modeled as a graph $G(V, E)$ where V is the set of vertices (nodes of WSN) and E is the set of edges (communication links between the nodes). *Graph*

theoretic algorithms use results from graph theory and develop algorithms to solve various difficult problems. Graph theoretic clustering algorithms, similarly, assume that the underlying network is modeled as a graph and provide clusters of the network using this property.

Various algorithms are proposed in literature for clustering in WSNs. LEACH [1] assumes cluster heads consume uniform energy and it provides random rotation of cluster heads. Cluster heads in HEED [2] are elected using a probabilistic method which is based on the residual energy of a node and the number of its neighbors. EECS [3], EEUC [4], and EEMC [5] also employ clustering in WSNs for communication purposes and these systems do not use graph theoretic approaches for clustering.

[6] is one of the only algorithms that performs graph theoretic clustering in sensor networks. Examples of graph theoretic clustering algorithms for various networks are given in [7, 8, 9, 10, 11, 12, 13, 14]. After the network is modeled as a graph then a completely new algorithm may be proposed or distributed versions

of centralized algorithms may be designed. Existing graph theoretic clustering algorithms generally aim at constructing a backbone of cluster heads directed to the sink node where selection of strong communication channels and cluster quality is not considered when forming the backbone. However, selection of links with higher communication capabilities reduces the total energy consumption at the message transmissions in multi-hop sensor network topologies [15].

A matching in a graph G is a set of non-loop edges with no shared endpoints. The vertices incident to the edges of a matching M are saturated by M . A perfect or maximum matching in a graph is a matching that saturates every vertex [16]. If the graph is weighted and we are looking for the maximum selected edge weight than the problem is called weighted matching. Matching has numerous applications such as chemical structure analysis, pattern recognition and machine learning [17].

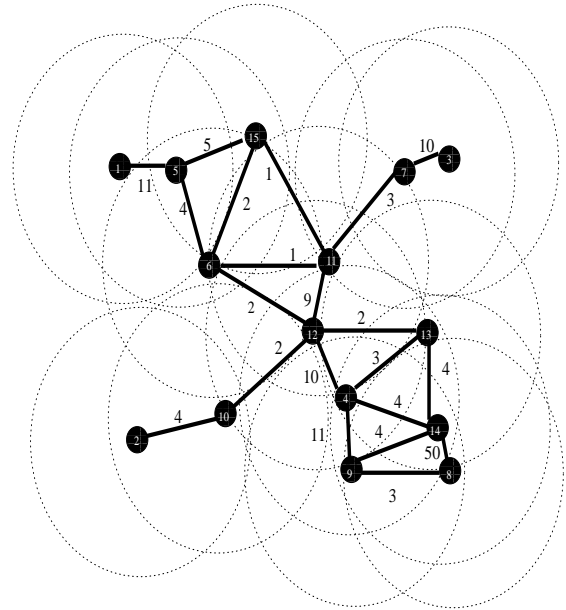
In this study, we propose two weighted matching based algorithms for clustering and backbone formation in sensor networks. The algorithms try to select heavy edges to provide communication over energy efficient communication links for reliable and efficient communication. In the first algorithm, clusters are enlarged in a synchronous manner where algorithm is executed in rounds and a certain task is performed at each round. The second algorithm is the enhanced version of the first algorithm where backbone formation is also provided along with the application of energy efficient techniques to reduce total number of messages. To the best of our knowledge, these two algorithms are the first algorithms in literature that perform graph theoretic clustering using matching in any type of network. Furthermore, they consider quality of the communication links to provide reliable communication among and within the clusters which is overlooked by the existing clustering algorithms. Lastly, they are fully distributed in nature making them suitable for large scale applications such as sensor networks.

The rest of this paper is organized as follows. In Section 2, the network model and clustering problem is described and the related work is surveyed in Section 3. The proposed algorithms are described in Section 4 and Section 5. The detailed analysis of the algorithms are given in Section 6 and the results of performance tests are presented in Section 7. Possible sensor network applications that may employ the proposed algorithms and extensions to the algorithms are given in Section 8. Conclusions and future works are given in Section 9. Pseudo-codes of the algorithms are given for reference in Section 10.

2. BACKGROUND

2.1. Network Model

The following assumptions are made about the network as in [2, 6, 7, 9, 12]:



2.2. The Clustering Problem

Clustering is a basic method to group similar objects from a whole set of objects. The number of groups must be controllable, the size of the groups should be balanced in any clustering scheme and also the operation of grouping should not be costly. In networks, clustering is performed to simply partition the whole network into subnetworks to ease communication tasks. Backbone formation is the construction of the virtual path of cluster heads to provide the relaying of sensed data to the sink. The aims of the proposed approaches for clustering varies. For example, the algorithm in [6] targets to guarantee that each ordinary node is connected to a single cluster head to construct disjoint clusters, whereas the algorithm in [22] argues that overlapping clusters is also a good idea for fault tolerance, node localization, and topology control. The algorithm in [12] produces clusters with star topology in which the cluster head is at the center of the star, on the other hand multi-hop clusters are produced in [6]. Clustering and backbone formation objectives for sensor networks can be listed as follows:

1. Nodes may initiate the clustering and backbone formation operation at any time locally. Hence, these operations should be distributed and asynchronous.
2. In clustering and backbone formation operations, strong communication links should be selected for ease of communication.
3. The cluster heads are the servers of their cluster members. They collect sensed data from their members to process, aggregate, filter and route this data to the sink. A cluster head may consume its energy very fast if the cluster is overcrowded. On the other hand, construction of clusters with very small sizes may not be cost effective. Conclusively, number and sizes of clusters should be controllable by the algorithms to adjust them to the required values.
4. The clustering algorithms should be independent from the underlying protocols as much as possible to interface to various MAC and physical layer standards such as in [18, 19, 23, 24, 25, 26].
5. The algorithms should be efficient in terms of time and message complexity to provide low energy consumptions of sensor networks.
6. Overlapping clusters may provide fault tolerance but sending sensed data by a member node to more than one cluster head may degrade the energy of nodes. The clusters may not be overlapping if fault tolerance is provided by other methods. The fault tolerance methods proposed are discussed in Section 8.

3. RELATED WORK

3.1. Graph Theoretic Clustering Algorithms

Graph theoretic clustering algorithms for ad hoc networks can mainly be classified as dominating set based and spanning tree based. 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 [16]. If the nodes in the dominating set is connected then the dominating set is called connected dominating set (CDS). A two-phased CDS algorithm is proposed in [9], in which initially each vertex marks itself as dominator due to some predefined rules. Dai [10], Nanuvala [11], and Cokuslu [12] added extra heuristics to Wu's algorithm to reduce the size of the connected dominating set. Yan [13] proposed an energy efficient dominating set based construction by using the nodes' residual power.

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 [27]. Banerjee and Khuller [14] proposed a protocol based on a spanning tree for hierarchical routing in wireless networks. In their scheme, a cluster is a subset of vertices whose induced graph is connected. Erciyes [6] proposed a distributed spanning tree based clustering algorithm (DSTA) for sensor networks. The depth parameter is provided by the algorithm to adjust the diameter of the clusters. The sink periodically sends *PARENT*(*nhops*) message to its neighbors to reinitiate the operation. Each node sends the *PARENT*((*nhops* + 1) mod *depth*) message to its neighbors upon first reception of the *PARENT*(*nhops*) message. The recipient of the message with *nhops* = 0 are the *SUBROOTS*, *nhops* < *depth* are the *INTERMEDIATE* nodes, *nhops* = *depth* are *LEAF* nodes.

3.2. Graph Matching Algorithms

Graph matching algorithms in the literature can be classified into 5 types:

1. Maximum matching algorithms for undirected graphs [28],
2. Maximum weighted matching algorithms for undirected graphs [29],
3. Matching and weighted matching algorithms for bipartite graphs [30],
4. Matching and weighted matching algorithms for trees [31],
5. Matching algorithms for directed graphs [32].

Maximum weighted matching algorithms (type 2) choose the edges to maximize the total weight in this model. Thus type 2 algorithms are the basis of our clustering operation. Maximum weighted matching is in polynomial complexity set (P) and approximation algorithms are proposed for this problem to provide more efficient solutions. Maximum

weighted graph matching algorithms can be centralized or distributed. The centralized algorithms are: Gabow's maximum weighted matching algorithm [33], Avis's algorithm with 1/2 approximation ratio [34], Preis's algorithm with 1/2 approximation ratio [35], Drake's algorithm with $2/3-\epsilon$ approximation ratio [36], Petite's algorithm with $2/3-\epsilon$ approximation ratio [37]. The distributed algorithms are: Uehara's algorithm with $1/\Delta$ (Δ is the maximum nodal degree) approximation ratio [38], Wattenhofer's algorithm with 1/5 approximation ratio [39], Hoepman's algorithm with 1/2 approximation ratio [29], Lotker's algorithm with 1/2 approximation ratio [40] and Nieberg's algorithm with $1-\epsilon$ approximation ratio [41].

Hoepman's algorithm is the distributed version of Preis's centralized algorithm. In Hoepman's algorithm, it is assumed that all of the nodes know their neighbors and the weights of the edges incident to them. Each node sends a *request* message to its candidate. The candidate is the neighbor node that is connected on the heaviest edge. If two of the nodes both receive *request* from each other, the edge connecting them is selected and they will be matched. If a node sends a *request* message to a matched node, matched node will reply with a *drop* message. Each node uses two sets: the set of neighbors (N) and the set of requested neighbors (R). A node deletes the source of *drop* message in its (N).

4. MATCHING BASED SYNCHRONOUS CLUSTERING ALGORITHM

4.1. General Idea

We propose the matching based synchronous clustering algorithm (MASC) which constructs clusters by using maximum weighted matching. One of the goals of the algorithm is the selection of the strong communication links in the undirected weighted graph model to maximize the intra-cluster communication and to focus on clustering quality. MASC works in rounds and each round starts after the previous one is completed by all cooperating nodes. At each round, each cluster head tries to merge with the adjacent cluster over the maximum weight edge connecting them. Merging of clusters at each round reduces the number of dummy clusters that frequently occur in DS based clustering algorithms. Our algorithm is an extended version of Hoepman's distributed matching algorithm [29] adapted for sensor networks. Hoepman's algorithm requires only local message exchanges providing scalability on sensor networks. Although the approximation ratio of the algorithm is 1/2, its approximation ratios are very high for randomly and uniformly generated sensor networks as shown in Section 7.1. Even though the approximation ratio of Nieberg's algorithm is $1-\epsilon$, it requires construction of a maximal independent set which will result in high time and message complexities for our implementation, thus it is not suitable as a basis for our clustering algorithm.

4.2. Description

We assume that the nodes are time synchronized in addition to the assumptions made in Section 2.1 to be able to start the rounds at the same time. The local algorithm consists of sending messages over adjoining links, waiting for incoming messages and processing messages. The finite state machine (FSM) of the algorithm is shown in Fig. 2, its pseudo-codes are given in Alg. 1 and Alg. 2.

In a clustering session, a node firstly sends a *REQUEST* message along the strongest link to a destination node. The destination node which is connected to the originator node with the strongest link is called the *candidate* of the originator node. If the originator node is already the candidate of the destination node, destination node will reply with *REQUEST_ACK* and clustering operation will be completed. Otherwise if destination node is matched with another node, it will reply with a *DROP* message and will search for the new candidates. This mechanism is similar to Hoepman's algorithm; our extension and contribution is to classify nodes as cluster heads and cluster members and giving them different responsibilities for constructing higher level clusters. The states of the algorithm are as follows:

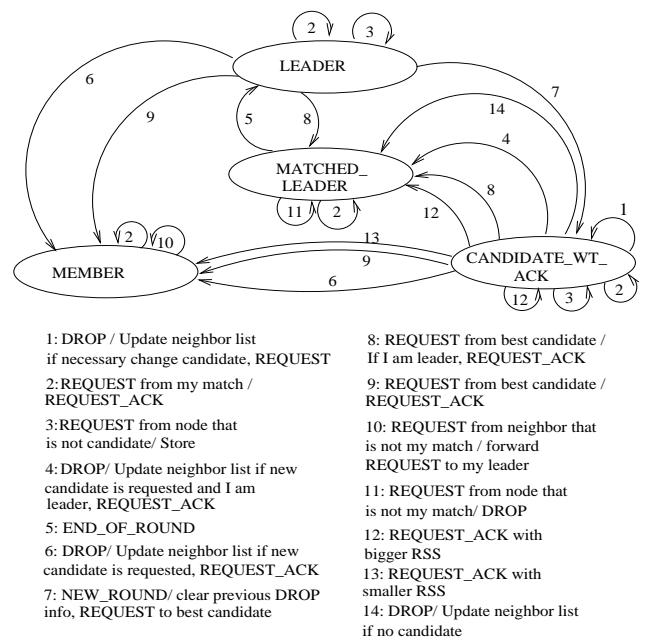


FIGURE 2. Finite State Machine of MASC.

- **LEADER:** *LEADER* is the cluster head of its cluster. All the nodes are in *LEADER* state at the beginning of the algorithm execution. If a *LEADER* receives a *REQUEST* from its candidate, it replies with a *REQUEST_ACK* message to maintain a new cluster. In this clustering scheme, one of the nodes will make a state transition to *MATCHED_LEADER* state and the other node

will be the *MEMBER* of the new cluster. The selection is made by executing the following procedure:

1. Let RSS_i be the second largest RSS value of the edge outgoing from the cluster of node $_i$ and RSS_j for node $_j$. The id of the node $_i$ is id_i , node $_j$ is id_j .
2. The node with the largest RSS is the new leader and makes a state transition to *MATCHED_LEADER* state, the other node is the member.
3. If RSS values are equal, the node with the bigger id is the new leader.

We use this leader selection heuristic, since node connected on the edge with second largest RSS will be the next candidate. *LEADER* node knows all of the neighbors of its cluster members and updates its list for each clustering session by getting the neighbor list from the previous *LEADER* nodes which participated in these operations.

- *MATCHED_LEADER*: A node in this state rejects the clustering requests by sending a *DROP* message after which it makes a state transition to *LEADER* state.
- *MEMBER*: Cluster member nodes are in *MEMBER* state and they forward the requests to their cluster head.
- *CANDIDATE_WT_ACK*: At the start of a round, each *LEADER* node sends a *REQUEST* message to its candidate and makes a state transition to *CANDIDATE_WT_ACK* state. If a *DROP* message is received, new candidate is selected. If no other candidate exists, node will make a transition to *MATCHED_LEADER* state.

We also provide a modified version of MASC called MOD-MASC which uses node id's in matching instead of edge weights to provide a reference for tests to evaluate the difference in choosing the maximum weight edge or a random edge. MOD-MASC chooses the neighbor with the smallest id for clustering.

4.3. An Example Operation

Assume a WSN with 20 nodes in which also the id of the nodes and the weights of the edges connecting them are given, as shown in Fig. 3. The tracing of an example clustering operation for 3 rounds using MASC is as follows. In the first round, a *NEW_ROUND_INTERRUPT* occurs in node 13. Node 13 is in initially *LEADER* state then it makes a state transition to *CANDIDATE_WT_ACK* state and sends a *REQUEST* message to its candidate, node 0. The state of the node 0 before receiving the *REQUEST* messages is *LEADER* and its candidate is node 13, so it replies with a *REQUEST_ACK* message. The second largest RSS value of the node 13 is node 5 and node 0 is node 7 so node 0 makes a state transition

to *MATCHED_LEADER* state and node 13 makes a state transition to *MEMBER* state. At the same, time node 11 sends a *REQUEST* message to node 0. The request of the node 11 is rejected with a *DROP* message from node 0. Node 14 and node 3, node 4 and node 5, node 12 and node 9, node 15 and node 2, node 10 and node 17, node 6 and node 1, node 11 and node 7 makes clustering operations in the first round similar to the above operation explained. On the other hand, all requests of node 16, node 8, node 19, and node 20 are dropped by their candidates and they can not find any cluster to join in the first round.

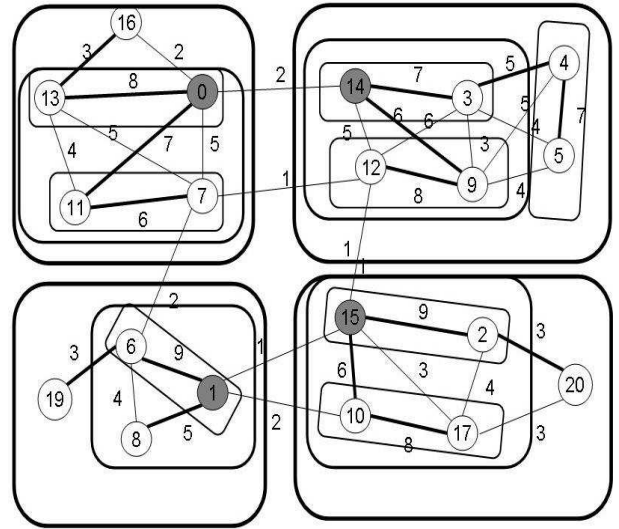


FIGURE 3. An Example Operation.

In the beginning of the second round, node 12 sends a *REQUEST* message to node 3. Node 3 that is in *MEMBER* state receives the *REQUEST* message and forwards to its *LEADER*, node 14. Node 14 replies to the *REQUEST* message with *REQUEST_ACK*, since the edge connecting the two clusters is the strongest link between them. After *REQUEST_ACK* is received by node 12, the nodes of the new cluster are: {14, 3, 12, 9}. At the end of the second round, the other clusters are formed as: {13, 0, 11, 7}, {6, 1, 8}, {15, 10, 2, 17}. The clustering operations are similar in the third round and at the end of the algorithm, the clusters produced by the algorithm are as follows: {14, 3, 12, 9, 4, 5}, {6, 1, 8, 19}, {13, 0, 16, 11, 7}, {15, 10, 2, 17, 20}. The cluster head nodes of each cluster are filled with gray and all of the matched edges are drawn bold as shown in Fig. 3.

5. MATCHING BASED CLUSTERING AND BACKBONE FORMATION ALGORITHM

5.1. General Idea

MASC uses the weighted graph matching as the method to select the strong communication links for clustering, different than the other graph theoretic algorithms.

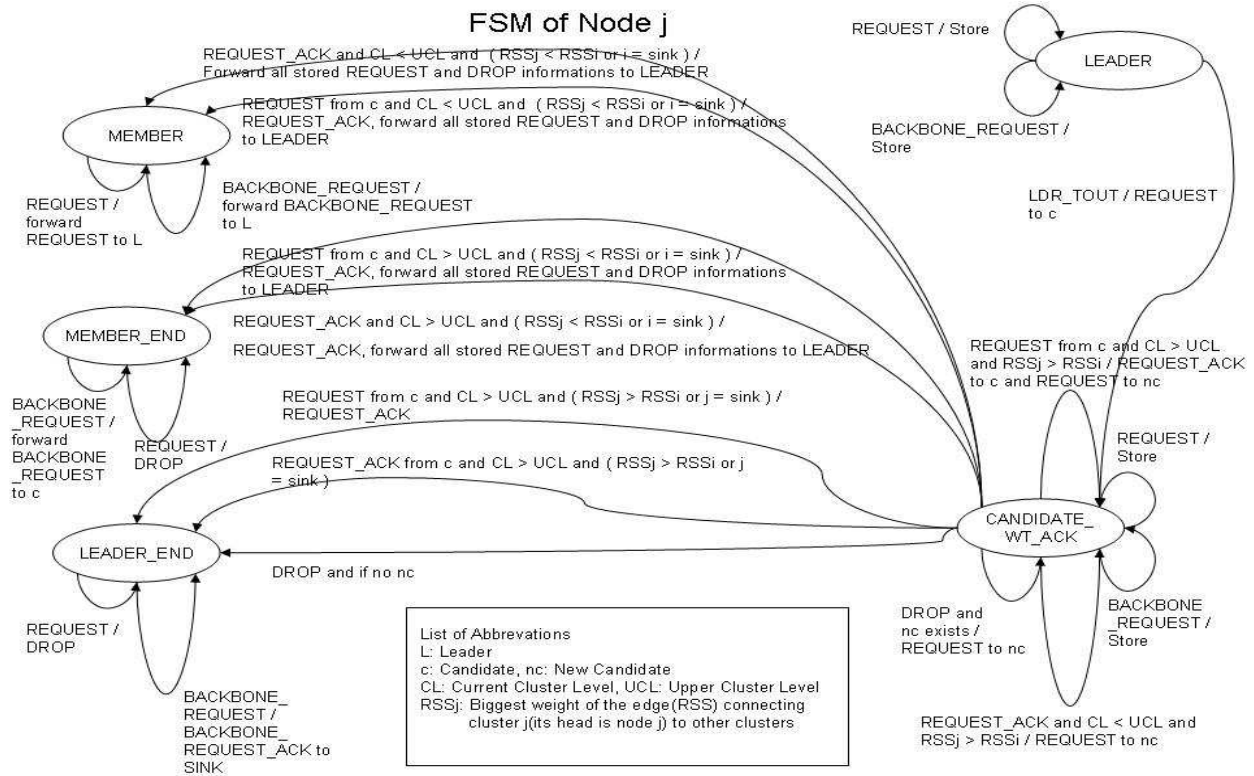


FIGURE 4. MCUBA's Finite State Machine for Node $_j$.

Also the number of clusters can be controlled by the number of rounds different than the DS based clustering algorithms [9, 10, 11, 12, 13]. The advantages of MASC over other graph theoretical approaches is described in 6 and 7. On the other hand, MASC lacks some important design issues:

- The nodes are assumed to be time synchronized. To maintain this situation, the nodes should implement either a time synchronization protocol [42] or a synchronizer middleware [43].
- The energy consumption of the algorithm execution is greater than the proposed graph theoretical approaches as shown in Section 6 and 7.
- The backbone formation is left to an upper layer protocol.

We propose a matching based asynchronous clustering and backbone formation algorithm (MCUBA) to decrease the disadvantages of MASC as much as possible while preserving its advantages. The MCUBA is based on weighted matching same as in MASC, but in addition to MASC, MCUBA provides fully asynchronous clustering and backbone formation. Also, MCUBA includes methods for reducing energy consumption.

5.2. Asynchronous Cluster and Backbone Formation

MCUBA algorithm constructs clusters and a backbone concurrently and asynchronously. The FSMs for the

MCUBA are given in Fig. 4 and Fig. 5 and the detailed pseudo-codes are given in Alg. 3, Alg. 4, Alg. 5, and Alg. 6. FSM in Fig. 4 is used by all of the nodes except the sink node which employs the FSM shown in Fig. 5. *LEADER*, *CANDIDATE_WT_ACK* and *MEMBER* states of MCUBA are similar with those states of MASC's with small modifications. Initially all the nodes except sink node executing MCUBA are in *LEADER* state. After a *LDR_TOUT* occurs at a node, it sends a *REQUEST* message to its candidate and makes a state transition to the *CANDIDATE_WT_ACK* state. If the destination node's candidate is the originator node, then the destination node sends a *REQUEST_ACK* message to the originator node. After the originator node receives *REQUEST_ACK* message, it can make a state transition to either *LEADER*, *LEADER_END*, *MEMBER* and *MEMBER_END* state. The basic message flow and leader election procedure of MCUBA is similar to MASC's.

To control the cluster levels, a parameter named *upper_level* is introduced. Before the execution of the algorithm, each node's *upper_level* parameter is set to the same value. If a leader node's cluster level reaches the *upper_level*, it makes a state transition to the *LEADER_END* state. A node in *LEADER_END* state drops all of the clustering requests. The same procedure is applied to the member nodes. A node in *MEMBER_END* state drops all clustering requests

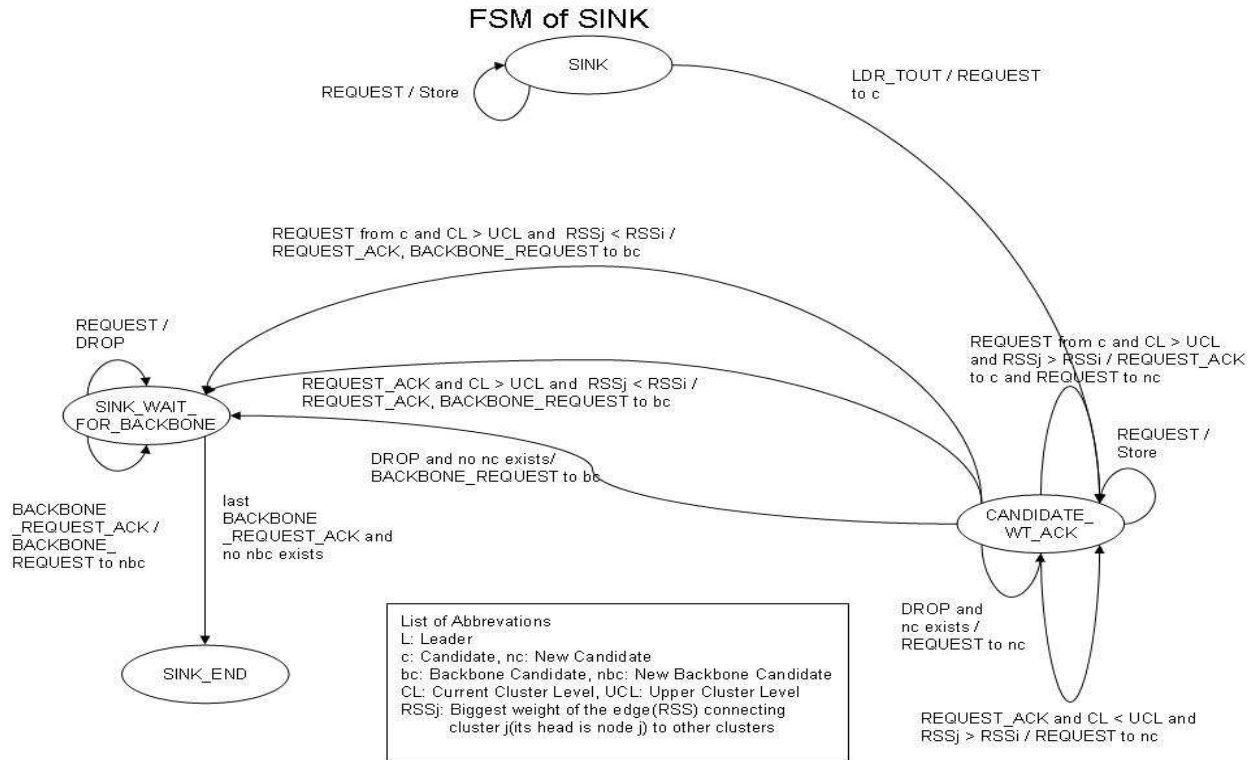


FIGURE 5. MCUBA's Finite State Machine for Sink.

without forwarding them to its leader. This approach provides balanced clusters and eliminates unnecessary forwarding of the *REQUEST* and *DROP* messages.

When a node receives a *REQUEST* message from one of its neighbors other than its candidate, it stores this request. After a leader node completes its clustering session, it searches for a new candidate unless it reaches the *upper_level*. The leader node's new candidate might be the source of the previous *REQUEST* message. In this case, leader node replies with a *REQUEST_ACK* message. This approach is different from MASC's. In MASC, a node in *MATCHED_LEADER* state drops all requests.

The backbone formation of the MCUBA is similar to the clustering operation. The main idea behind the backbone formation is combining all of the clusters as a super cluster. The backbone formation is managed by the sink node. The FSM of the sink node is shown in Fig. 5. Initially, sink node participates in the clustering operation similar to the other ordinary nodes. Different than other nodes, sink node is always the leader in clustering sessions. After the cluster of the sink node reaches the *upper_level*, it starts backbone formation. In each backbone formation session, sink node selects the locally heaviest edge outgoing from its cluster. When the cluster level of sink node reaches the *upper_level*, sink node makes a state transition from *CANDIDATE_WT_ACK* state to *SINK_WT_FOR_BACKBONE* state by sending

a *BACKBONE_REQUEST* message along the outgoing edge with the heaviest weight. This message is forwarded to the leader of the destination cluster. If this cluster's level already has reached the *upper_level*, the leader of this cluster replies with a *BACKBONE_REQUEST_ACK*. Otherwise, *BACKBONE_REQUEST* is stored in this cluster head until the cluster level reaches *upper_level*. If the cluster head changes, the old cluster head forwards the stored *BACKBONE_REQUEST* message to the new leader. When sink node receives a *BACKBONE_REQUEST_ACK*, it merges the source cluster and its own cluster then continues the same operation described below until all clusters are merged.

5.3. Reducing Energy Consumption

Overhearing occurs when a node receives a packet that is not destined to itself [23]. Although overhearing causes energy consumption, it may be useful to gather neighborhood information, reliable routing and distributed query processing. To maintain reliable message transmission, *ACK* messages are sent for each successful message transfer. Overhearing technique is applied to reduce the number of *ACK* messages to decrease the energy consumption. In Fig. 6, a sensor network with 3 nodes is shown. In the left side of Fig. 6, node A sends a message to node C without using overhearing. In the first step, node A sends its message

to node B. Node B sends the *ACK* message to node A after it successfully receives the message of node A. In the third step, node B routes the message of node A to node C and node C sends the *ACK* message to node A as the last step. If we use overhearing technique, the total number of steps reduces to three, since the routed message by node B is also used as an *ACK* message.

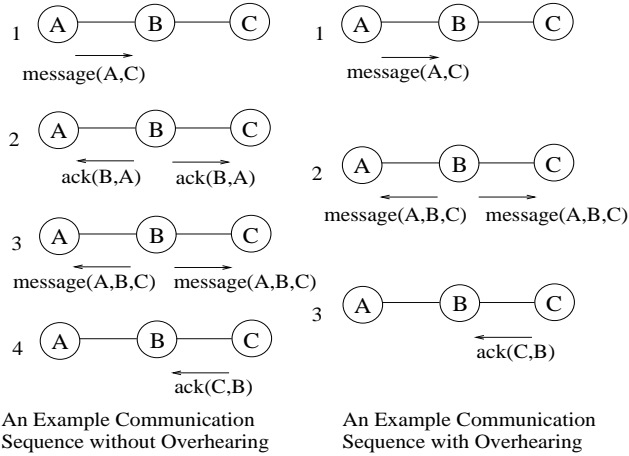


FIGURE 6. Overhearing Technique.

5.4. An Example Operation

An example operation of MCUBA with *upper_level*=4 on a network with 20 nodes is shown in Fig. 7. Each node's id and the weights of the edges connecting the nodes are given. Node 0 is the sink node executing the algorithm in Fig. 5, other nodes are the ordinary nodes executing the algorithm in Fig. 4. Initially all ordinary nodes are in *LEADER* state, sink node is in *SINK* state. Node 0's candidate is node 16, since the heaviest edge outgoing from node 0 is the edge which connects node 0 to node 16. Node 0 sends a *REQUEST* message to node 16. Node 16's candidate is node 0, so node 16 replies with a *REQUEST_ACK* message to node 0 and they construct a new cluster with level 2. Since node 0 is the sink node, it is chosen as the leader of this cluster. Concurrently, node 5 and node 19 perform a clustering operation similar to this operation. The second largest RSS value of node 5 is 8, node 19's is 5 as shown in Fig. 7, thus node 5 is chosen as the leader. Node 9 and node 2, node 1 and node 14, node 6 and node 18, node 11 and node 4, node 12 and node 8, node 13 and node 7 make clustering operations similar to the explained operations above.

Node 17 had sent a *REQUEST* message to node 2 while node 2 was making a clustering operation with node 9. Since node 2's candidate was not node 17, node 2 did not send a *REQUEST* message but it recorded the *REQUEST* of node 17. After node 2 completes its clustering session with node 9, it sends a *REQUEST_ACK* message to node 17 since node 17 becomes the new candidate of node 2. The new

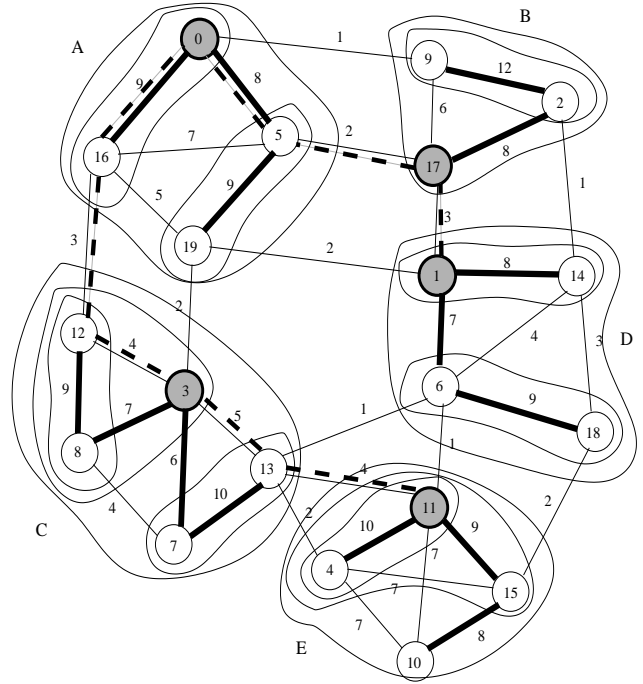


FIGURE 7. An Example Operation of MCUBA.

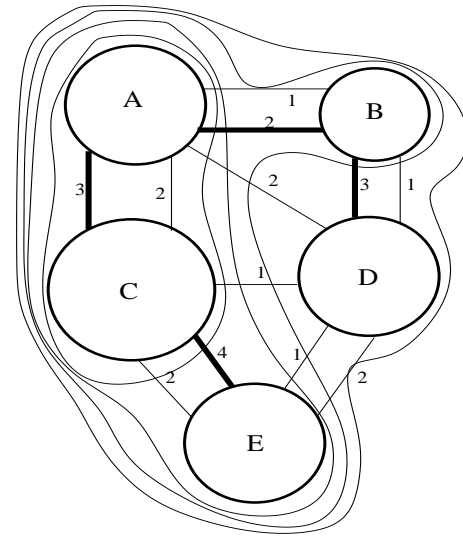


FIGURE 8. An Example Backbone Formation of MCUBA.

cluster is formed as: {17, 2, 9}, and the new leader of this cluster is node 17. The other clusters with level 3 are constructed as: {3, 8, 12} and {11, 4, 15}. Concurrently, node 1 sends a *REQUEST* message to node 6. The candidate of node 6 is node 1, so node 6 sends a *REQUEST_ACK* message to node 1 and cluster D is formed: {1, 14, 6, 18} as shown in Fig. 7. The leader of this cluster, node 1, makes a transition to *LEADER_END* state since its cluster level reaches the *upper_level*. The nodes in *LEADER_END* state reject all clustering requests. Node 0 merges its cluster with node 5's cluster and cluster A becomes: {0,

5, 19, 16} as shown in Fig. 7. The cluster level of node 0 exceeds the *upper_level*, node 0 makes a transition to *SINK_WT_FOR_BACKBONE* state, and starts backbone formation operation. Node 0 sends a *BACKBONE_REQUEST* message to node 12, since node 12 is connected to node 0's cluster on the heaviest edge as seen in Fig. 7. After node 12 receives *BACKBONE_REQUEST* message, it forwards the message to its leader, node 3, since node 12 is a cluster member. Node 3 records the received *BACKBONE_REQUEST* message and do not respond since it's clustering operation is not finished.

Node 17 receives *DROP* messages from all its candidates since the levels of cluster A and cluster D reaches *upper_level* and their leaders reject all clustering requests. Because of this situation, node 17 makes a transition to *LEADER_END* state. The cluster B is formed as: {17, 12, 9}. At the same time, the clusters of node 3 and node 7, and the clusters of node 11 and node 10 are merged. The cluster C is formed as: {3, 12, 8, 13, 7}, and the cluster D is produced as: {3, 12, 8, 13, 7} shown in Fig. 7. Selected edges for communication during the clustering sessions are shown as bold in Fig. 7.

The leader of cluster C, node 3, sends a *BACKBONE_REQUEST_ACK* message in reply to recorded *BACKBONE_REQUEST* after level of cluster C reaches *upper_level*. After node 0 receives the *BACKBONE_REQUEST_ACK* message of node 3, cluster C is merged with the backbone. Node 0 sends a *BACKBONE_REQUEST* message to node 11 along the heaviest edge which connects backbone to other clusters. Node 11 is in *LEADER_END* state and replies with a *BACKBONE_REQUEST_ACK* message after it receives *BACKBONE_REQUEST* message from node 0. When node 0 receives the *BACKBONE_REQUEST_ACK* from node 11, cluster E is merged with the backbone. Similarly cluster B and cluster D are merged into the backbone. The iterations of the backbone formation is depicted in Fig. 8. The bold lines in Fig. 8 are the selected edges between clusters for the backbone formation. All the selected edges of backbone are shown with the dashed lines in Fig. 7.

6. ANALYSIS

6.1. Proof of Correctness

OBSERVATION 1. In MASC and MCUBA, in a clustering session, source node sends a *REQUEST* message, destination node either replies with a *DROP* or *REQUEST_ACK* message.

OBSERVATION 2. In MASC, the nodes in *MEMBER* state are the cluster members, the nodes in *MATCHED_LEADER* state are the cluster heads. In MCUBA, the nodes in *MEMBER* and

MEMBER_END state are the cluster members, the nodes in *LEADER_END* state are the cluster heads and the node in *SINK_END* state is the sink node.

OBSERVATION 3. In a backbone formation session of MCUBA, source node sends a *BACKBONE_REQUEST* message, destination node replies with a *BACKBONE_REQUEST_ACK* message.

LEMMA 6.1. *In MASC and MCUBA, if a node sends a REQUEST message to its candidate, it receives a DROP or REQUEST_ACK message from its candidate.*

Proof. We first prove the MASC by cases. The cases explained below are also shown in Fig. 2.

- *Case 1:* The candidate is in either *LEADER* or *CANDIDATE_WT_ACK* state. It sends a *REQUEST_ACK* message.
- *Case 2:* The candidate is in *MATCHED_LEADER* state. It sends a *DROP* message.
- *Case 3:* The candidate is in *MEMBER* state. It forwards the message to its leader. Due to Observation 2, the leader can not be in *MEMBER* state. Thus we jump to the Case 1 or Case 2.

□

LEMMA 6.2. *In MASC no state transitions are possible for a node in MEMBER state and no state transitions before the end of the round is possible for a node in MATCHED_LEADER state. In MCUBA, no state transitions are possible for a node in MEMBER, MEMBER_END, LEADER_END, or SINK_END state.*

Proof. In MASC, a node makes a state transition from *MATCHED_LEADER* state to *LEADER* state when *END_OF_ROUND* occurs. No other state transitions are possible for MASC and MCUBA as seen in FSMs in Fig. 2, Fig. 4, Fig. 5, with the pseudocodes in Alg. 1, Alg. 2, Alg. 3, Alg. 4, Alg. 5, and Alg. 6. □

LEMMA 6.3. *In MCUBA, if a source node sends a BACKBONE_REQUEST message, it will be replied with a BACKBONE_REQUEST_ACK message.*

Proof. The destination is either a cluster head, member or a node that continues clustering operation. We prove the lemma by cases.

- *Case 1:* The node is in *LEADER_END* state. In this state, node replies with a *BACKBONE_REQUEST_ACK* message.
- *Case 2:* The node is in *MEMBER* or *MEMBER_END* state. The node forwards the *BACKBONE_REQUEST* message to its leader.

- *Case 3:* The node is in *LEADER* or *CANDIDATE_WT_ACK* state. It records the *BACKBONE_REQUEST* message. If the node makes a state transition to *MEMBER* or *MEMBER_END* state, it forwards the previously recorded *BACKBONE_REQUEST_ACK* message to its leader. Else if the node makes a state transition to *LEADER_END* state, it sends a *BACKBONE_REQUEST_ACK* message.
- *Case 4:* The node is in *SINK*, *SINK_END*, or *SINK_WT_FOR_BACKBONE* state. If a node is in one of these states, it is not possible to receive *BACKBONE_REQUEST_ACK* since there is only one sink node. As seen in FSMs in Fig. 4 and Fig. 5, with the pseudo-codes in Alg. 3, Alg. 4, Alg. 5, and Alg. 6. this case is not possible.

□

THEOREM 6.1. *After the execution of MASC, each node is a cluster head or a member node belonging to a cluster.*

Proof. We prove by contradiction. If the theorem is valid then according to Observation 2 the node is in *MEMBER* or *MATCHED_LEADER* state. We assume the contrary that a node is in *CANDIDATE_WT_ACK* or *LEADER* state after the execution of the algorithm. We investigate each cases:

- *Case 1:* The node is in *LEADER* state. It must not receive any *REQUEST* message from its candidate or send any *REQUEST* messages to its candidate. Because if one of these events occurs, node makes a state transition. If node does not receive any *REQUEST* message, a *LEADER_TOUT* occurs, it makes a state transition to *CANDIDATE_WT_ACK* state and sends a *REQUEST* message to its candidate. According to Lemma 6.1, if it receives a *REQUEST* message from its candidate it sends a *REQUEST_ACK* message and makes a state transition to *MEMBER* or *MATCHED_LEADER* state. If the node does not have any candidate, a *LEADER_TOUT* occurs and it makes a state transition to *MATCHED_LEADER* state.
- *Case 2:* The node is in *CANDIDATE_WT_ACK* state. The node must not receive any message from its candidate. Because if it receives *REQUEST* or *REQUEST_ACK* from its candidate, it makes a state transition to *MATCHED_LEADER* or *MEMBER* state. If the node receives *DROP* message, it makes a state transition to *MATCHED_LEADER* state or sends a *REQUEST* message to its candidate. According

to Lemma 6.1, if a node sends *REQUEST* message to its candidate, it receives *REQUEST_ACK* or *DROP* message.

All nodes make a state transition to *MEMBER* or *MATCHED_LEADER* state. Lemma 6.2 shows that state transitions from *MEMBER* or *MATCHED_LEADER* states in a round are not possible. We contradict with our assumption. □

THEOREM 6.2. *After the execution of MCUBA, a node is the sink node, a cluster head, or a member node belonging to a cluster.*

Proof. The proof is the same as in Theorem 6.1. We assume the contrary. Thus the node is either in *LEADER*, *CANDIDATE_WT_ACK*, *SINK*, or *SINK_WT_FOR_BACKBONE* state. Each case is investigated below:

- *Case 1:* The node is in *LEADER* state. The node must not receive any messages from its candidate or send any messages to its candidate. If the node receives any message from its candidate, it makes a state transition to the *MEMBER*, *MEMBER_END* or *LEADER_END* state. If the node sends a *REQUEST* message to its candidate, it makes a state transition to *LEADER* state. If the node has no candidate, it makes a state transition to *LEADER_END* state.
- *Case 2:* The node is in *SINK* state. The possibilities are the same as in case 1. The node makes a state transition to *CANDIDATE_WT_ACK*, *SINK_WT_FOR_BACKBONE* or *SINK_END* state.
- *Case 3:* The node is in *CANDIDATE_WT_ACK* state. The node must not receive any messages. Due to Lemma 6.1, node receives a *DROP* or *REQUEST_ACK* message.
- *Case 4:* The node is in *SINK_WT_FOR_BACKBONE* state. The node must not receive a *BACKBONE_REQUEST_ACK* message for each *BACKBONE_REQUEST* message. But due to Lemma 6.3, each *BACKBONE_REQUEST* message is replied.

In all cases, each node makes state transition to *MEMBER*, *MEMBER_END*, *LEADER_END* or *SINK_END* states. Due to Lemma 6.2, state transitions from these states are not possible. We contradict with our assumption. □

COROLLARY 6.1. *MCUBA and MASC are free from deadlock and starvation.*

Proof. We assume the contrary. We assume that MCUBA and MASC are not free from deadlock and starvation. Due to Observation 1, in a clustering

session of MCUBA and MASC, source node sends a *REQUEST* message, destination node replies with a *DROP* or *REQUEST_ACK* message. There must be a problem in this message flow. But it is proven in Lemma 6.1, Lemma 6.2, and Theorem 6.1, that all message flows related to clustering are successful. Due to Observation 3, in a backbone formation session of MCUBA, each received *BACKBONE_REQUEST* is replied by a *BACKBONE_REQUEST_ACK* message. If there is a deadlock or starvation then there must be a problem in this flow. But it is proven in Lemma 6.3 and Theorem 6.2 that all message flows related to backbone formation are successful. We contradict with our assumption. \square

6.2. Clustering Quality

THEOREM 6.3. *The upper bound of the cluster level for MASC with r rounds is 2^r .*

Proof. The proof is by induction.

- *Base Step:* When $r=1$, the algorithm runs same with Hoepman's distributed matching. Thus upper bound is 2.
- *Induction step:* We assume that after r^{th} round ends, the upper bound is 2^r . At each round, only one clustering operation is permitted to a cluster head. Because of this, at $(r+1)^{th}$ round, the upper bound is $2^r + 2^r = 2^{r+1}$.

\square

THEOREM 6.4. *The upper bound of the cluster level for MCUBA with $s=upper_level$ is $2(s-1)$.*

Proof. The leader node continues clustering operation until it reaches s . So that the upper bound of the cluster level occurs when two clusters with $s-1$ levels combine. \square

6.3. Selected Edge Weights

In this section we analyze the lower bounds of the total selected edges for clustering and backbone formation operations in MASC and MCUBA. The list of variables used in this section is given below:

- W : The total selected edge weight for the clustering operation by using perfect weighted matching.
- B : The total selected edge weight for the backbone formation operation by using perfect weighted matching.
- A : The total selected edge weight by MASC with r rounds.
- S : The total selected edge weight for the clustering operation by MCUBA with $upper_level=2^r$ and $s=upper_level$.
- C : The total selected edge weight for the backbone formation operation by MCUBA with $upper_level=2^r$ and $s=upper_level$.

THEOREM 6.5. *The lower bound of A is:*

$$A \geq W \sum_{i=1}^r (1/2)^i.$$

Proof. We prove by induction.

- *Base Step:* When $r=1$, the algorithm turns to Hoepman's distributed matching. Thus $A \geq W 1/2$ is true.

- *Induction step:* We assume that for $r=n$, $A \geq W \sum_{i=1}^n (1/2)^i$ holds. For $r=n+1$, selected edge weights will be $1/2$ of the remaining at the worst case.

$$A \geq W \sum_{i=1}^n (1/2)^i + (W - W \sum_{i=1}^n (1/2)^i) 1/2$$

$$A \geq W \sum_{i=1}^n (1/2)^i + W(1 - (1 - 1/2^n)) 1/2$$

$$A \geq W \sum_{i=1}^n (1/2)^i + W 1/2^{n+1}$$

$$A \geq W \sum_{i=1}^{n+1} (1/2)^i$$

\square

THEOREM 6.6. *The $S+C$ is between:*

$$W \sum_{i=1}^s (1/2)^i \leq S + C \leq W + B$$

Proof. We prove by induction.

- *Base Step:* For $r=1$, $s=2$. The upper bound of cluster level produced by MCUBA with $s=2$ is $2s-2$ from Theorem 6.4. In this case, MASC and MCUBA creates same clusters. From Theorem 6.5, the selected edge weights are:

$$W 1/2 \leq (S=A) \leq W+B$$

$$\text{For } 0 \leq C \leq B$$

$$W 1/2 \leq (S=A) \leq S+C \leq W+B$$

$$W 1/2 \leq S+C \leq W+B$$

- *Induction step:* We assume that for $r=2^n$, the theorem is true. When $r=2^{n+1}$, $upper_level$ of MCUBA is $2 \cdot 2^{n+1} - 2 = 2^{n+2} - 2$. In this case, the selected edge weights for clustering:

$$S=W \sum_{i=1}^{2^{n+2}-2} (1/2)^i$$

$$W \sum_{i=1}^{n+1} (1/2)^i \leq W \sum_{i=1}^{2^{n+2}-2} (1/2)^i + B \leq W+B$$

$$\text{For } n \geq 0, W \sum_{i=1}^{n+1} (1/2)^i \leq W \sum_{i=1}^{2^{n+2}-1} (1/2)^i \text{ then}$$

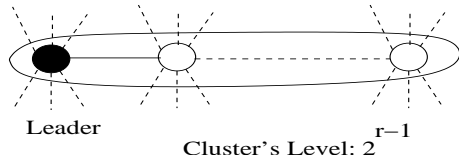


FIGURE 9. Topology for Worst Case Message Complexity of MASC.

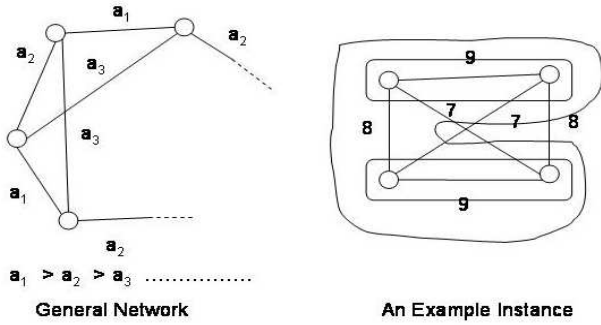


FIGURE 10. Topology for Best Case Message Complexity of MCUBA.

$$W \sum_{i=1}^n (1/2)^i \leq E + B \leq W+B$$

□

6.4. Message and Time Complexities

THEOREM 6.7. *The message complexity of MASC with r rounds is $O(\Delta 2^r)$ per node. The lower bound for message complexity is $\Omega(r)$ per node.*

Proof. In the worst case, the leader of the cluster with 2^{r-1} level is rejected by all neighbors of all nodes belonging to the same cluster. This case is depicted in Fig. 9. The total message transfer in this case is:

$$\frac{2(\Delta-1)+\dots+2(2^{r-1}(\Delta-1)+(\Delta-1))}{2^{r-1}}$$

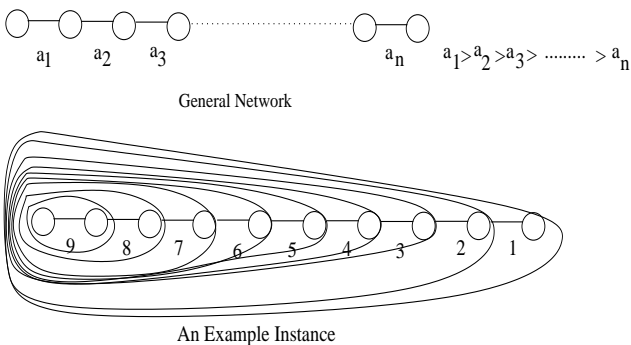


FIGURE 11. Topology for Worst Case Time Complexity of MCUBA.

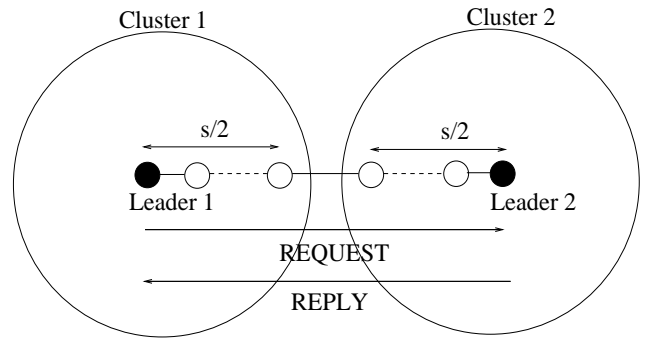


FIGURE 12. An Average Clustering Session.

$$\frac{2(\Delta-1)(1+2+\dots+2^{r-1}+1)}{2^{r-1}}$$

$$\frac{(\Delta-1)(2^{r-1}+1)(2^{r-2}+1)}{2^{r-1}}$$

$$\left(\frac{\Delta 2^r}{4} + \Delta + \frac{\Delta}{2} + \frac{2\Delta}{2^r} - 2^{r-2} - \frac{3}{2} - \frac{2}{2^r} \right) \in O(\Delta 2^r)$$

The lower bound occurs when each node only sends one *REQUEST* message and receives a *REPLY* message. In this case only $2r$ message exchanges are done. Thus, the lower bound for the message complexity is $\Omega(r)$. □

THEOREM 6.8. *The time complexity of the MASC with r rounds is $O(\Delta 2^r r)$.*

Proof. The worst case for time complexity is the same as the worst case for the message complexity as seen in Fig. 9. Thus the time complexity of a round is $O(r\Delta)$. The time complexity for r rounds is $O(\Delta 2^r r)$. □

THEOREM 6.9. *The message complexity of MCUBA with $s=upper_level$ has a lower bound of $\Omega(2)$ per node, has an upper bound of $O(\Delta s)$.*

Proof. The worst case is same with MASC as shown in Fig. 9. But the cluster level depends on s . Besides that, each node may send n/s backbone formation message. The calculations for the worst case are given below:

$$\frac{2(\Delta-1)+\dots+2(2s-2(\Delta-1)+(\Delta-1))}{2s-2} + \frac{n}{sn}$$

$$\frac{2(\Delta-1)(1+2+\dots+2s-1)}{2s-2} + \frac{1}{s}$$

$$\frac{(\Delta-1)(2s-1)(2s)}{(2s-2)} + \frac{1}{s}$$

$$\left(\frac{\Delta 2s^2 - \Delta 2s - 2s^2 + 2s}{2s-2} + \frac{1}{s} \right) \in O(\Delta s)$$

The best case occurs in the complete graph K_n in Fig. 10. n node exchange a *REQUEST* and a *REPLY* message. After then, $n/2$ node exchange a *REQUEST* and a *REPLY* message. The operation goes in this way. The equation is given below:

$$\frac{n(1+1/2+1/4+\dots)}{n} \leq 2$$

□

THEOREM 6.10. *The time complexity of the MCUBA has a lower bound of $\Omega(\log(n))$ and upper bound of $O(n)$.*

Proof. We consider clustering and backbone formation together. The best case occurs when clusters with level 1 are connected to form clusters with level 2, clusters with level 2 are connected to form clusters with level 4, and the operation continues similarly. Fig. 10 shows the sample network for the lower bound. We need $\Omega(\log(n))$ time for the best case. The worst case occurs when a cluster starts from a single node and ends with level n . In this case, $n-1$ iterations are done. So that the worst case time complexity is $O(n)$. Fig. 11 shows the worst case iterations. □

6.5. Reductions in Energy Consumption

THEOREM 6.11. *Assume that the nodes are distributed uniformly and randomly. Also assume that the total energy consumption of the nodes are mainly caused by message transmissions (other energy consumptions than radio are negligible). Overhearing reduces at least 40 % of the energy consumed at an average clustering session of MCUBA with upper_level ≥ 8 .*

Proof. Let s be the upper_level. Since cluster level varies between 1 and $2s-2$ with the same probability, the average value of the cluster level is s . In this case, the average value of the cluster diameter is $s/2$. Fig. 12 shows the average clustering session between two clusters. Let a *CONTROL_ACK* message be a low level acknowledgement for reliable transmission. Without overhearing, for each message transmission of MCUBA, a *CONTROL_ACK* message should sent so that the total message transfers for an average clustering session without overhearing is (N):

$$N = (\text{REQUEST messages} + \text{REPLY messages} + \text{CONTROL_ACK messages})$$

$$N = ((s/2+1) + (s/2+1) + (s+2))$$

$$N = 2s+4$$

By using overhearing, there is no need for *CONTROL_ACK* messages except the two last messages which are not routed. So that the total number of messages with overhearing technique is (T):

$$T = ((s/2+1) + (s/2+1) + 2)$$

$$T = s+4$$

$$\frac{N-T}{T} = \frac{s}{2s+4}$$

$$\text{for } s \geq 8, \frac{s}{2s+4} \cdot 100 \geq 40 \%$$

□

7. PERFORMANCE EVALUATION

We firstly implemented Hoepman's distributed matching in *ns2* simulator version 2.31 to test its performance. To calculate the approximation ratio of Hoepman's algorithm, we implemented Gabow's maximum weighted algorithm [33]. After this, the proposed MASC and MCUBA algorithms were implemented in *ns2* simulator. Besides, we implemented a modified version of MASC (MOD-MASC), which uses node id's in matching instead of edge weights. MOD-MASC chooses the neighbor with the smallest id for clustering. MOD-MASC has the same time and message complexities as MASC, however its approximation ratio may vary significantly due to its random operation. To compare our algorithms with the existing graph theoretic approaches, we implemented DSTA and a dominating set based algorithm (DS). DSTA constructs clusters and ensures spanning tree formation rooted at the sink as described in Section 3.1. We set the *depth* parameter of DSTA to 3 in our experiments. DS is a connected dominating set based clustering algorithm, an algorithm similar to Wu's algorithm. Initially all nodes are white in DS algorithm. DS has two rules to find connected dominating set:

1. If the node has two unconnected neighbors it marks itself as black.
2. If the node's neighbors with greater id cover all neighbors of the node, node marks itself as white.

We generated randomly connected networks with 100 to 400 nodes that are uniformly distributed. IEEE 802.11 radio and MAC standards readily available in *ns2* simulator were chosen for lower layer protocols. Two way ground was used as the propagation model. The transmission power is 0.660 mW, the received power is 0.395 mW, and the communication range of a sensor node is 250 m. We measured the performance of the algorithms for average node degrees varying between 4,5 and 6. To vary degrees, different size of flat surface areas were chosen as shown in Table 1.

7.1. Tests for Hoepman's Algorithm

The approximation ratios of Hoepman's algorithm were measured against the varying number of nodes and node degrees. To find the approximation ratio, we derived a weighted graph, $G_w = (V, E_w)$, in which V is the set of nodes in simulation area and E is the set edges with RSS values connecting nodes. Gabow's algorithm is not distributed, it inputs G_w and outputs the maximum weighted matching set E_M . Let E_H be the matching found by Hoepman's algorithm, T_H be

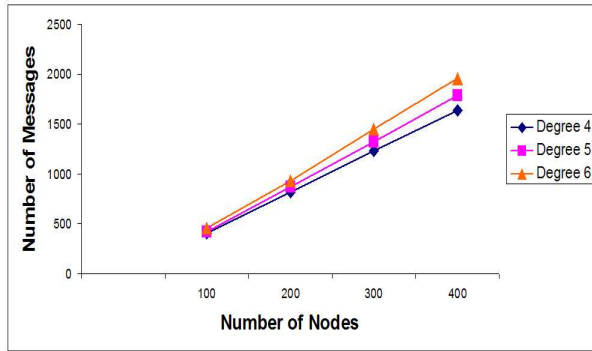


FIGURE 13. Number of Messages Used in Hoepman's Algorithm.

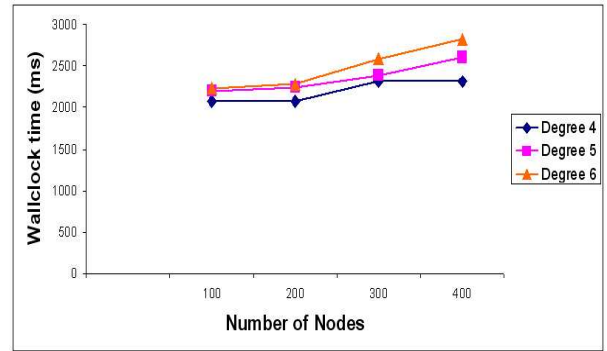


FIGURE 14. Wallclock time values of Hoepman's Algorithm.

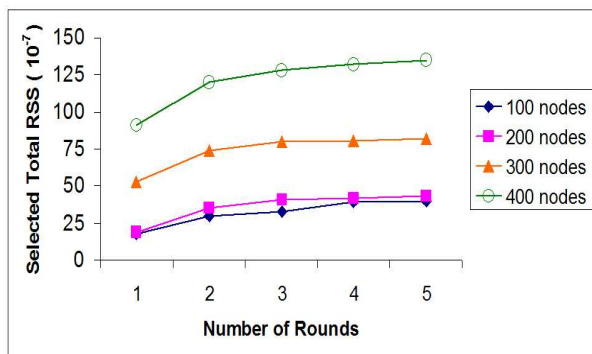


FIGURE 15. Total Selected Edge Weights by MASC.

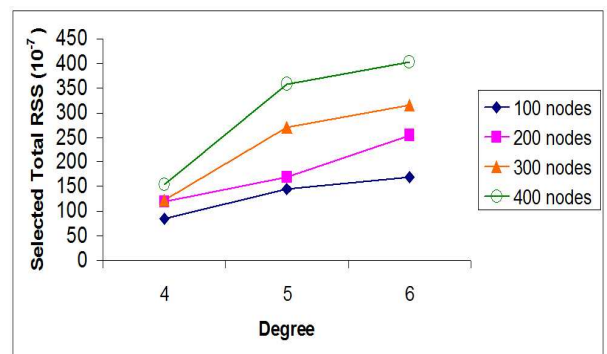


FIGURE 16. Total Selected Edge Weights by MCUBA.

TABLE 1. Size of Surface Areas ($X \times Y$ (m)).

Node Number/Degree	4	5	6
100	2700×1200	2520 ×1200	2340×1040
200	5100×1200	4760 ×1120	4420×1040
300	7800×1200	7280 ×1120	6760×1040
400	10200×1200	9520 ×1120	8840×1040

TABLE 2. Approximation Ratios of Hoepman's Algorithm.

Node Number/Degree	4	5	6
100	0.9986	0.9982	0.9977
200	0.9986	0.9983	0.9983
300	0.9992	0.9987	0.9984
400	0.9991	0.9988	0.9986

the total edge weights in E_H and T_M be the total edge weights in E_M , then the approximation ratio of the Hoepman's algorithm is T_H/T_M . Table 2 shows the approximation ratios of Hoepman's algorithm. The measured approximation ratio values are greater than 0.99 and are independent from node number and node degree variations. Although the worst case approximation ratio of the algorithm is 0.5 theoretically, for networks of randomly and uniformly dispersed sensor nodes, the approximation ratio is close to the maximum value of 1.

Secondly, we measured the total number of messages and wallclock time values of the Hoepman's algorithm. Fig. 13 shows the number of messages used in Hoepman's algorithm. Number of messages increase linearly against the number of nodes, and increase slightly against the node degree. A node, on the

average, sends four messages for distributed matching including the acknowledgment messages for reliable message transfer. The wallclock time values of the algorithm varies between 2s to 3s as seen in Fig. 14 and as with the total number of message measurements, wallclock time values are stable against the node number and average node degree. We can conclude from our measurements that Hoepman's algorithm has very high approximation ratios and it is scalable with regards to message and wallclock time measurements. Thus, Hoepman's algorithm is very suitable as the basis of our iterative clustering algorithms for sensor networks.

7.2. Selected Edge Weights

Selection of stronger links for clustering and backbone formation is an important criteria for our algorithms.

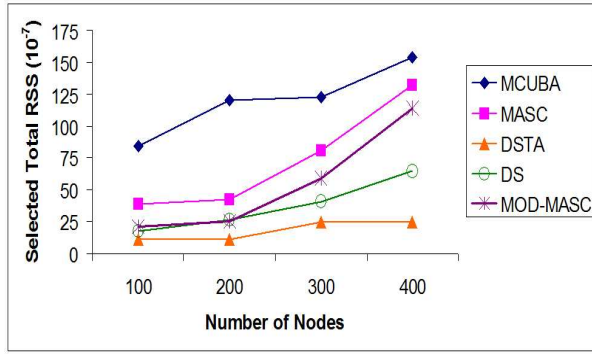


FIGURE 17. Total Selected Edge Weights by Algorithms.

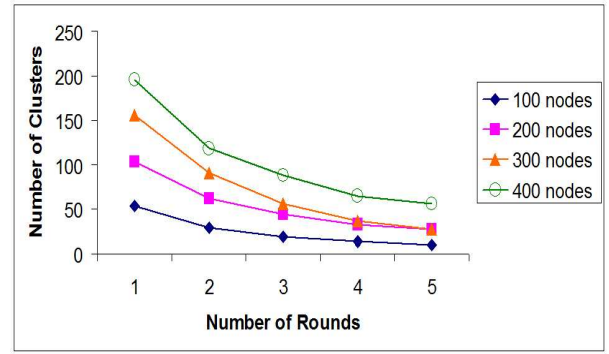


FIGURE 18. Number of Clusters Produced by MASC.

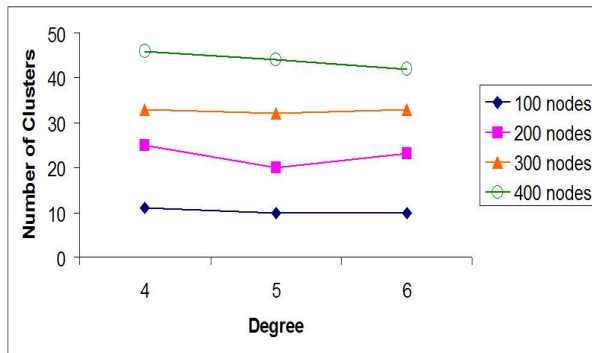


FIGURE 19. Number of Clusters Produced by MCUBA.

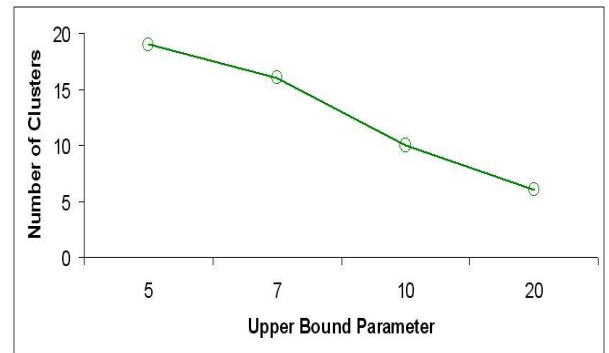


FIGURE 20. Clusters Produced by MCUBA against *upper_level*.

We measured the total selected edge weights of MASC and MCUBA. The edge weights were represented with the RSS values defined in IEEE 802.11 [19]. The weight of selected edges in r^{th} round are heavier than $(r + 1)^{th}$ round, so the slope of the function must be decreasing. As seen in Fig. 15, as the number of rounds and number of nodes increase, the selected total edge weight by MASC increases. As the number of rounds increases, the slope of the function decreases as shown in Fig. 15. We measured the total selected edge weights by MCUBA with *upper_level*=10. As the average node degree increases, although the percentage of strong links to weak links remains same, the total number of strong links increases. Since MCUBA chooses strong links instead of randomly choosing, the total selected edge weight increases as the number of nodes and average node degree increases as shown in Fig. 16.

We compared the total selected edge weights of the algorithms as seen in Fig. 17. We can classify the algorithms as: Weighted matching based algorithms (MCUBA and MASC), matching based algorithms (MOD-MASC), dominating set based algorithms (DS), and spanning tree based algorithms (DSTA). We implemented MCUBA with *upper_level*=10, MASC and MOD-MASC with 4 rounds, the cluster levels produced by the algorithms in this case are approximately equal. As seen in Fig. 17, weighted matching based and

matching based algorithms select stronger links than the dominating set based and spanning tree based approaches. It can be stated from here that, graph matching technique has a better performance than the existing graph theoretical approaches with regards to link selection. Weighted matching based algorithms target to maximize the edge weights, on the other hand id-based matching algorithms randomly chooses the edges. Due to this fact, as seen in Fig. 17, the total selected RSS values of MCUBA and MASC are higher than the MOD-MASC. MCUBA constructs clusters and forms backbone, on the other hand MASC only produces clusters. Because of this, the total edge weights selected by MCUBA for communication is higher than those selected by MASC as seen in Fig. 17. On the average, if the total edge weights selected by DSTA is W , the edge weights selected by DS is $2.3W$, MOD-MASC is $3.7W$, MASC is $4.6W$ and MCUBA is $7W$.

7.3. Clustering Quality

To evaluate quality of the produced clusters, we used two metrics: Number of clusters and the node count in clusters. The node count in a cluster is called the level of the cluster. The number of clusters must be controllable in a preferable clustering algorithm. In MASC, for each round, low level clusters merge to form

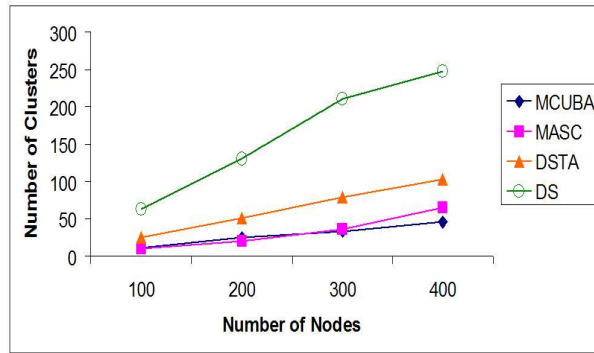


FIGURE 21. Number of Clusters Produced by Algorithms.

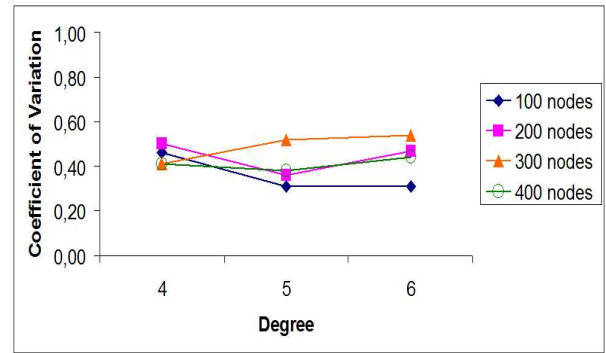


FIGURE 22. CV of Clusters Produced by MCUBA.

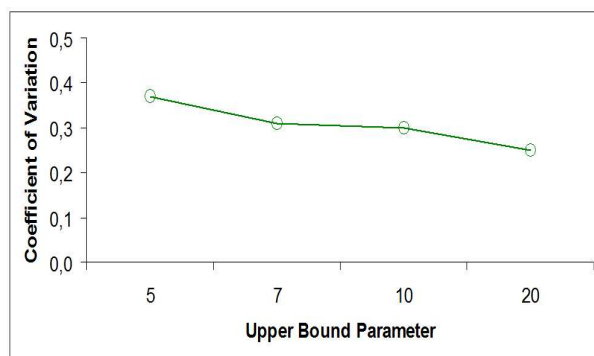


FIGURE 23. CV of Clusters Produced by MCUBA against *upper_level*.

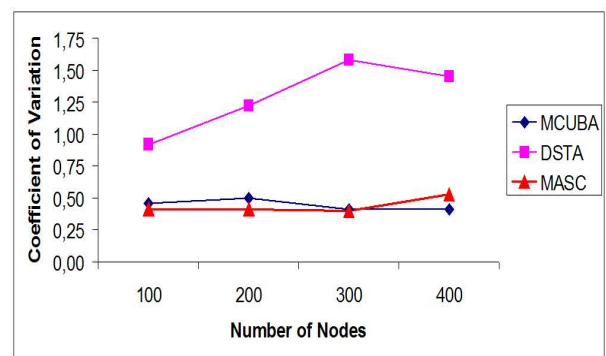


FIGURE 24. CV of Clusters Produced by Algorithms.

higher level clusters such that the number of clusters decrease as the number of rounds increase. As seen in Fig. 18, the number of clusters are approximately equal to the half of the number of nodes in the first round and cluster number decreases as the number of rounds increases. In MCUBA, the cluster levels can be adjusted by the *upper_level* parameter, so that the number of clusters is controllable. The number of clusters produced by MCUBA with *upper_level*=10 is measured against number of nodes and average node degree as shown in Fig. 19. The number of clusters produced by MCUBA is stable and controllable as seen in this figure. We also vary the *upper_level* of MCUBA between 5, 7, 10, and 20 to measure the effect of *upper_level* parameter as shown in Fig. 20. The number of produced clusters by MCUBA are 19, 16, 10 and 6 respectively. From these measurements, we can state that the MCUBA's *upper_level* parameter controls the number of clusters efficiently as expected.

A comparison of the number of clusters produced by algorithms is seen in Fig. 21. DS is restricted by the connected dominating set construction, thus number of clusters may not be controllable. On the other hand, the cluster levels are adjustable in DSTA with *depth* parameter. Among all of the implemented algorithms, DS has the worst performance as shown in Fig. 21 because number of clusters is more than

half of the nodes. DSTA with *depth*=3 performs well since the number of clusters increases linearly with a small slope as the number of nodes increases. MCUBA with *upper_level*=10 produces 46 clusters, MASC with 4 rounds produces 65 clusters, DSTA produces 102 clusters, DS produces 247 clusters for the sensor network with 400 nodes having average degree of 4. MCUBA and MASC perform very well since the number of clusters is stable against the number of nodes. They achieve the merging clustering operations to construct clusters with higher level, thus the number of clusters are controllable.

The second criteria of the clustering performance is the balancing of the clusters. Our balance metric is the coefficient of variation (CV) which is computed as *standard deviation / mean*. If $CV < 1$ is then the distribution is considered to be of low variance else it is of high variance. We measure the CV values of MCUBA against node degree and *upper_level* parameter. The CV values of MCUBA with *upper_level*=10 are stable against node degree and number of nodes as seen in Fig. 22. The average value of CV measurements of MCUBA with *upper_level*=10 is 0.43. We fix the average node degree to 4 and vary the *upper_level* of MCUBA between 5, 7, 10, and 20 to measure CV. The measured CV values are stable and vary between 0.40 and 0.25 as seen in Fig. 23. The comparison of CV

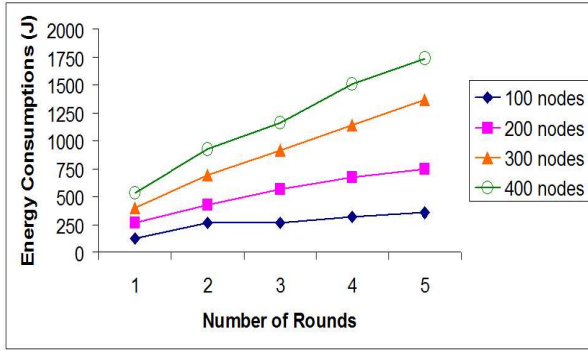


FIGURE 25. Energy Consumption of MASC.

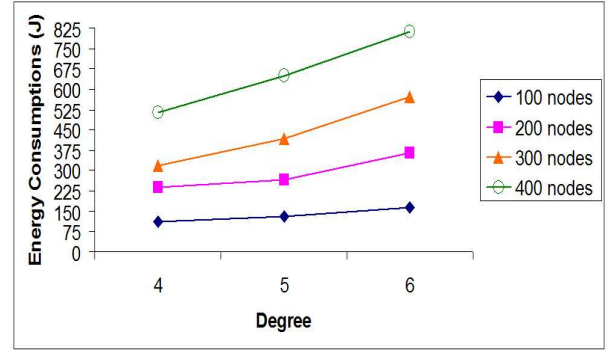


FIGURE 26. Energy Consumption of MCUBA.

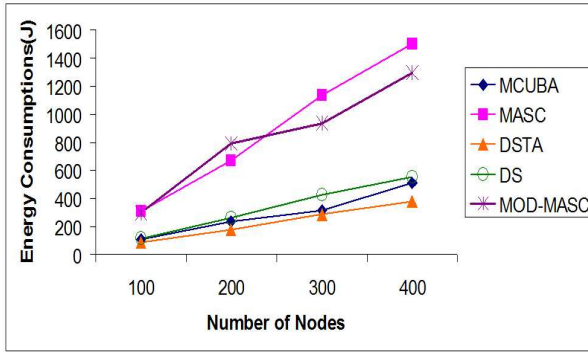


FIGURE 27. Energy Consumptions of Algorithms.

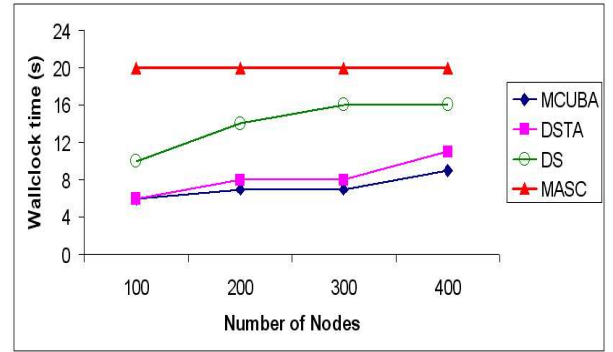


FIGURE 28. Wallclock Time of Algorithms

values measured by MASC with 4 rounds, MCUBA with $upper_level=10$ and DSTA with $depth=3$ is seen in Fig. 24. Since the number of clusters produced by DS is not controllable, we do not include DS in this comparison. The average CV values of MASC, MCUBA and DSTA are 0.44, 0.45, and 1.29 respectively.

7.4. Energy Consumptions

Energy efficiency is an important objective for WSN. We measured the energy consumption of the algorithms for clustering and backbone formation. It is assumed that the energy consumptions occur mostly by message transfers. The energy consumption of MASC increases linearly against the number of rounds and the number of nodes as shown in Fig. 25. Beside this, as shown in Fig. 26, the energy consumed by MCUBA with $upper_level=10$ increases linearly against the node degree and the number of nodes. We compared the energy consumption of all approaches as seen in Fig. 27. In DS and DSTA, nodes exchange less number of messages than those in MASC and MOD-MASC, so the energy consumptions of MASC and MOD-MASC are higher. The main reason of this energy consumption is the operation of merging low level clusters to construct high level clusters. To decrease this energy consumption, overhearing technique was applied in MCUBA. The effect of overhearing technique applied in MCUBA to reduce the energy consumption

can be seen in Fig. 27. The energy consumption of DS, DSTA and MCUBA perform similarly. On the average, DSTA consumes 0.90J, MCUBA consumes 1.16J, DS consumes 1.32J, MOD-MASC consumes 3.28J, MASC consumes 3.53J per node.

7.5. Wallclock Times

We measured the wallclock times of MCUBA with $upper_level=10$, MASC with 4 rounds, DSTA with $depth=3$ and DS. As seen in Fig. 14, at the worst case, the time for the distributed matching is 3s approximately. In our measurements, the time for a round of MASC is 5s in the worst case, so we set the round of MASC to 5s. Although the nodes are idle when they finish the execution of the MASC in a round before 5s, they must wait until the end of the round. Since nodes execute asynchronously in MCUBA, there is no waiting time for all nodes. The wallclock of MASC with 4 rounds is constant and equals 20s. In DSTA, a node forwards each received message to all neighbors, unless it is forwarded before. In DS, the nodes decide their states by learning the states of their neighbors. In this type of algorithms, collisions may frequently occur since each node may exchange messages with its neighbors in the same time intervals. To prevent these collisions, IEEE 802.11 MAC uses an exponential back-off timer so that the execution time of the algorithms increases. As seen in Fig. 28, MCUBA performs best

among the other approaches, DSTA also performs well whereas MASC performs worst due to waiting times.

8. DISCUSSIONS

In this section, we discuss the applicability and the extensions of our proposed algorithms. MCUBA and MASC are based on selecting strong communication links as described in the previous sections. We use the RSS values available as implemented in IEEE 802.11 standards on ns2 as the link quality metric. Other quality metrics than RSS may be used. Lal shows that in a stationary sensor network, it is possible to obtain a good estimate of the true cost metric based on only a few measurements [15]. Lal also states that the energy consumed on a link has a strong relationship with the state of the wireless channel between two communicating nodes. Therefore one of the design parameters for the energy efficient multi-hop routing is the link selection. We define a sensor network application to run on MCUBA, DS, and DSTA where each node senses events and sends this messages to the sink node. This is a typical requirement for the habitat monitoring application. The definitions for this application and the network parameters are given below:

- n : Number of nodes
- Δ : Average node degree
- T : The total time for the application
- p : Period time for the clustering algorithms
- E : Average energy dissipated in a message transfer.
- V : The total number of events sensed by the nodes
- s : MCUBA's *upper_level* parameter.

We assume that an event is routed from the event catcher node to the sink node in $\log(n)$ hops on the average. Generally, in sensor network applications, nodes may malfunction due to various reasons. Therefore to provide fault tolerance, clustering algorithms run periodically. The message complexity of MCUBA is $O(\Delta s)$, DS and DSTA is $O(\Delta)$. From our simulations, we found the average link quality of DSTA as W , DS as $2.3W$, MCUBA as $7W$ as shown in Section 7.2. Let α , β and γ be the number of retransmissions for a successful packet transmission needed by DSTA, DS, and MCUBA. From Lal's study [15], we may state that $\alpha < \beta < \gamma$ with regards to average link qualities. The total energy consumption of the sensor network includes energy consumptions of the clustering algorithm and the application with the retransmissions. The total energy consumptions of MCUBA (M), DS (D), and DSTA (T) are:

$$M = ((n\Delta s)\lfloor T/p \rfloor + \log(n)V)\alpha E \quad (1)$$

$$D = ((n\Delta)\lfloor T/p \rfloor + \log(n)V)\beta E \quad (2)$$

$$T = ((n\Delta)\lfloor T/p \rfloor + \log(n)V)\gamma E \quad (3)$$

It is obvious from Eq. 2 and Eq. 3, D is smaller than T . We now compare M and D . From Eq. 2 and Eq. 1, $D-M$ is:

$$D - M = ((n\Delta)\lfloor T/p(\beta - s\alpha) \rfloor + \log(n)V(\beta - \alpha))E \quad (4)$$

From Eq. 4, as the number of events increases MCUBA consumes less energy than DS. For $\beta > s\alpha$, MCUBA consumes less energy than DS for varied number of nodes, average node degrees and application lifetimes. The application designer should consider these parameters before the selection of the clustering algorithm.

MCUBA and MASC may run on various underlying sensor network protocols. Both algorithms only require *send*, *receive* primitives and link quality indicator. The *send* and *receive* primitives are important services required from MAC and physical layer [18, 19, 23, 24, 25]. IEEE 802.15.4 provides link quality indicator (LQI) which can be calculated by the combination of signal to noise ratio and the detected energy. In S-MAC and TRAMA, probe packets and overhearing method are used to obtain link quality [44]. B-MAC [25], M-MAC [24], WMEWMA [45] are the other underlying protocols for the multi-hop routing network layers which provide link quality estimation.

MCUBA is the enhanced version of the MASC. The total selected edge weights, cluster quality, time and message complexity of the MCUBA are shown to be better than MASC. Also the simulation results show that MCUBA performs better than MASC. In some clustering approaches, the backbone formation algorithm is located on top of the clustering algorithm [46] where the clusters are constructed first then the cluster heads form the backbone. MASC can be used in this manner as the underlying protocol of a backbone formation algorithm. For example, a ring backbone with balanced clusters can be constructed with MASC and backbone formation algorithm (BFA) proposed in [46] for sensor networks. The second approach for backbone construction is increasing the transmission ranges of the cluster heads [2]. After clustering is completed, cluster heads increase their transmission range for communicating with each other. This approach is also suitable for the MASC. The energy consumption of the MASC can be reduced by the overhearing technique as applied in MCUBA.

Cluster heads may consume their energy more than ordinary nodes since they are responsible for data processing, aggregation and routing messages to the sink. So rotation of the cluster heads may be provided by the clustering approaches. MCUBA and MASC can be easily extended for energy efficient rotating cluster head selection. The graph model can be extended to $G_W(V_W, E_W)$ in which not only edges but also vertices

have weights where the weight of a vertex is its initial energy. We may change the leader selection policy of the MASC and MCUBA using this model. The candidate with the largest weight may be chosen as the backup. This method provides energy efficient cluster head selection and can be implemented on MASC and MCUBA with minor modifications.

The periodic running of the clustering algorithms is not the only way for supporting fault tolerance. One other method is the selection of backup cluster heads [2]. MCUBA and MASC can be extended to provide backup cluster heads. The FSM in Fig. 29 shows the additions needed to provide fault tolerance. After a node makes a state transition to *LEADER_END* state, it chooses a backup cluster head and sends *YOU_ARE_BACKUP* to it. Simply, the neighbor node in the same cluster and with the greatest energy may be chosen as the cluster head. If a node in *MEMBER* or *MEMBER_END* state receives *YOU_ARE_BACKUP* message, it makes a state transition to *BACKUP_LEADER* and the backup polls its cluster head by sending *POLL_LEADER* messages periodically and makes a state transition to *WT_LEADER* state. When the cluster head in *LEADER_END* state receives the *POLL_LEADER* message, it replies with a *HEARTBEAT* message. If the cluster head is not responding, backup cluster head tries to send *POLL_LEADER* message for *poll_number* times defined by the user. When backup realizes that the cluster head malfunctions, a *LEADER_DEAD* event occurs in backup and makes a state transition to *LEADER_END* state by sending a *YOU_ARE_BACKUP* message to the new backup. The backup can also malfunction. In this case, a *BACKUP_TOUT* occurs in cluster head, it selects a new backup and sends a *YOU_ARE_BACKUP* message to the new backup. This approach is good when one of the cluster head or cluster member malfunctions. If both of them malfunction, new techniques should be employed. To tolerate faults occurring in ordinary nodes, extended routing tables can be designed. In extended routing tables, a node may store more than one entry for each destination. This information may be obtained during message exchanges and overhearing. On the other hand, since this method causes significant memory space consumption, an optimization must be provided.

9. CONCLUSIONS

We proposed two new graph theoretic algorithms for clustering and backbone formation in WSNs. Our original idea is to use the weighted matching for clustering operation in order to select the edges with the heavy weights. We used Hoepman's distributed weighted matching algorithm as the basis of our proposed algorithms. Hoepman's algorithm is implemented and shown to be scalable for sensor

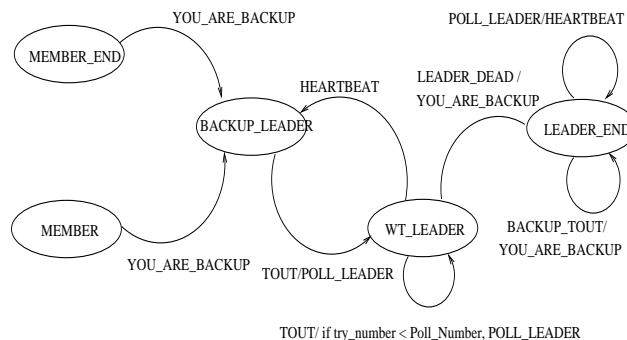


FIGURE 29. Finite State Machine for Providing Fault Tolerance.

networks in terms of total number of messages and time elapsed. Also, the measured approximation ratios for this algorithm using simulations are greater than 99 % of the maximum value which makes this algorithm very suitable for our clustering algorithm basis. We showed the design of the proposed algorithms, MASC and MCUBA, by FSM and pseudo-codes. We gave the analysis for the time and message complexity of the algorithms. Also, the lower bounds for the total selected edge quality are given with the upper bounds for the cluster level theoretically. Both algorithms are implemented on simulation environment and the results obtained show that our proposed algorithms select stronger links, produce more evenly distributed and controllable number of clusters than the other graph theoretical approaches. The energy consumed in MASC is reduced in MCUBA by applying overhearing technique. We discussed and analyzed that selection of strong links between nodes for clustering and backbone formation may provide energy efficient multi-hop routing for applications that may run on the infrastructure created by our proposed algorithms. Fault tolerance and energy efficient cluster head rotation extensions are also discussed for the proposed algorithms.

REFERENCES

- [1] Heinzelman, W., Chandrakasan, A., and Balakrishnan, H. (2002) An Application-Specific Protocol Architecture for Wireless Microsensor Networks. *IEEE Trans. on Wireless Comm.*, 1(4), 660 - 670.
- [2] Younis, O. and Fahmy, S. (2004) Distributed Clustering in Ad-hoc Sensor Networks: A Hybrid, Energy-Efficient Approach. *Proceedings of INFOCOM 04, Hong Kong, 7-11 March*, pp. 629-640, IEEE, Washington.
- [3] Ye, M., Li, C., Chen, G. and Wu, J. (2006) An Energy Efficient Clustering Scheme in Wireless Sensor Networks. *Elsevier Ad Hoc and Sensor Wireless Networks*, 3, 99-119.
- [4] Li, C., Ye, M., Chen, G. and Wu, J. (2005) An Energy-Efficient Unequal Clustering Mechanism for Wireless Sensor Networks. *Proceedings of MASS 05, Washington, 7-10 November*, pp. 604-612, IEEE,

- Washington.
- [5] Jin, Y., Wang, L., Kim, Y., and Yang, X. (2007) EEMC: An Energy-Efficient Multi-Level Clustering Algorithm for Large-Scale Wireless Sensor Networks. *Elsevier Computer Networks*, 52(3), 542-562.
 - [6] Erciyes, K., Ozsoyeller, D. and Dagdeviren, O. (2008) Distributed Algorithms to Form Cluster based Spanning Trees in Wireless Sensor Networks. *Proceedings of ICCS 08, Krakow, 23-25 June, LNCS 5101*, pp. 519-528, Springer-Verlag, Berlin.
 - [7] Chen, Y. P. and Liestman, A. L. (2002) Approximating Minimum Size Weakly-Connected Dominating Sets for Clustering Mobile Ad Hoc Networks. *Proceedings of MOBICOM 02, Atalanta, 23-28 September*, pp. 165-172, ACM Press, New York.
 - [8] Stojmenovic, I., Seddigh, M. and Zunic, J., (2001) Dominating Sets and Neighbor Elimination-based Broadcasting Algorithms in Wireless Networks. *IEEE Trans. on Parallel and Dist. Sys.*, 13, 14-25.
 - [9] Wu, J., and Li, H. (1999) A Dominating-Set-Based Routing Scheme in Ad Hoc Wireless Networks. *Telecommunication Systems*, 3, 63-84.
 - [10] Dai, F. and Wu, J. (2004) An Extended Localized Algorithm for Connected Dominating Set Formation in Ad Hoc Wireless Networks. *IEEE Trans. On Parallel and Dist. Sys.*, 15(10), 908-920.
 - [11] Nanuvala, N. (2006) An Enhanced Algorithm to Find Dominating Set Nodes in Ad Hoc Wireless Networks. Master of Science Thesis in the College of Arts and Science, Georgia State University.
 - [12] Cokuslu, D., Erciyes, K. and Dagdeviren, O. (2006) A Dominating Set Based Clustering Algorithm for Mobile Ad hoc Networks. *Proceedings of ICCSA 06, Glasgow, 8-11 May, LNCS 3991*, pp. 571-578, Springer-Verlag, Berlin.
 - [13] Yan, X., Sun, Y. and Wang, Y. (2003) A Heuristic Algorithm for Minimum Connected Dominating Set with Maximal Weight in Ad Hoc Networks. *Proceedings of GCC 03, Shanghai, 7-10 December, LNCS 3032*, pp. 719-722, Springer-Verlag, Berlin.
 - [14] Tech. Report CS-TR-4103 (2001) A Clustering Scheme for Hierarchical Routing in Wireless Networks. University of Maryland, Annapolis, Maryland, USA.
 - [15] Lal, D., Manjeshwar, A., Herrmann, F., Uysal-Biyikoglu, E. and Keshavarzian, A. (2003) Measurement and Characterization of Link Quality Metrics in Energy Constrained Wireless Sensor Networks. *Proceedings of GLOBECOM 03, San Francisco, 1-5 December, 1*, pp. 446-452, IEEE, Washington.
 - [16] West, D. (2001) *Introduction to Graph Theory*. Second edition, Prentice Hall, New Jersey.
 - [17] Bunke, H. (2000) *Graph matching: Theoretical foundations, algorithms, and applications*. *Proceedings of VI 03, Montreal, 11-13 June*, pp. 82-88, IEEE, Washington.
 - [18] IEEE Std 802.15.4TM-2003 (2003), IEEE Standard for Information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements-Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs), IEEE, Washington.
 - [19] IEEE 802.11 Standard Working Group (2007), Draft Standard for Information Technology - Telecommunications and Information Exchange Between Systems - LAN/MAN Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE P802.11-REVma/D9.0, IEEE, Washington.
 - [20] Kuhn, F., Moscibroda, T. and Wattenhofer, R. (2004) Unit Disk Graph Approximation. *Proceedings of DIALM-POMC 04, Philadelphia, 1 October*, pp. 17-23, ACM, New York.
 - [21] Clark, B. N., Colbourn, C. J. and Johnson, D. S. (1990) Unit Disk graphs. *Discrete Mathematics*, 86(1-3), 165-177.
 - [22] Youssef, A., Younis, M. F., Youssef, M. and Agrawala, A. (2007) Establishing Overlapped Multihop Clusters in Wireless Sensor Networks. *Int. J. Sen. Netw.*, 2, 108-117.
 - [23] Ye, W., Heidemann, J. and Estrin, D. (2004) Networking, Medium access Control with Coordinated Adaptive Sleeping for Wireless Sensor Networks. *IEEE/ACM Transactions*, 12(3), 493-506.
 - [24] Choi, L., Lee, S. H. and Choi, H. (2009) M-MAC: Mobility-Based Link Management Protocol for Mobile Sensor Networks. *Proceedings of of STFSSD 09, Tokyo, 17 March*, pp. 210-214. IEEE, Washington.
 - [25] Polastre, J., Hill, J. and Culler, D. (2004) Versatile Low Power Media Access for Wireless Sensor Networks. *Proceedings of SenSys 04, Baltimore, 3-5 November*, pp. 95-107, ACM, New York.
 - [26] Rajendran, V., Obraczka, K. and Garcia-Luna-Aceves, J. J. (2003). Energy-efficient Collision-free Medium Access Control for Wireless Sensor Networks. *Proceedings of SENSYS 03, Los Angeles, 5-7 November*, pp. 181-192. ACM, New York.
 - [27] Grimaldi, R. P., (1999) *Discrete and Combinatorial Mathematics. An Applied Introduction*. Addison Wesley Longman, New York.
 - [28] Czygrinow, A. and Hanckowiak, M. (2006) Distributed Algorithms for Weighted Problems in Sparse Graphs. *Journal of Discrete Algorithms*, 4(4), 588-607.
 - [29] cs.DC/0410047 (2004) Simple Distributed Weighted Matchings.
 - [30] Chattopadhyay, S., Higham, L. and Seyffarth, K. (2002), Dynamic and Self-stabilizing Distributed Matching. *Proceedings of PODC 02, Montemery, 21-24 July*, pp. 290-297, ACM, New York.
 - [31] Karaata, M. and Saleh, K. (2000) A Distributed Self Stabilizing Algorithm for Finding Maximal Matching. *Computer Systems Science and Engineering*, 3, 175-180.
 - [32] Hsu, S.-H., Hsu, C.-C., Lin, S.-S. and Lin, F.-C (2004). A Multi-Channel MAC Protocol Using Maximal Matching for Ad Hoc Networks. *Proceedings of ICDCSW 04, Tokyo, 23-24 March, 7*, pp. 505-510, IEEE, Washington.
 - [33] Gabow, H. (1990) Data Structures for Weighted Matching and Nearest Common Ancestors with Linking. *Proceedings of SODA 90, Vancouver, 23-25 January*, pp. 434-443, ACM, New York.
 - [34] Avis, D. (1983) A Survey of Heuristics for the Weighted Matching Problem. *Networks*, 3(4), pp. 475-493.

- [35] Preis, R. (1999) Linear Time 1/2-Approximation Algorithm for Maximum Weighted Matching in General Graphs. *ACM Transactions on Algorithms*, 1(1), pp. 107-122.
- [36] Drake, D. and Hougardy, S. (2003) Improved Linear Time Approximation Algorithms for Weighted Matchings. *Proceedings of APPROX 03*, Princeton, 24-26 August, LNCS 2764, pp. 14-23, Springer-Verlag, Berlin.
- [37] Pettie S. and Sanders, P. (2004) A Simple Linear Time $2/3-\epsilon$ Approximation for Maximum Weight Matching. *Information Processing Letters*, 91(6), pp. 271-276.
- [38] Uehara, R. and Chen, Z. (2000) Parallel Approximation Algorithms for Maximum Weighted Matching in General Graphs. *Information Processing Letters*, 76, pp. 13-17.
- [39] Wattenhofer, M. and Wattenhofer, R. (2004) Distributed Weighted Matching. *Proceedings of DISC 04*, Amsterdam, 4-7 October, LNCS 3274, pp. 335-348, Springer-Verlag, Berlin.
- [40] Lotker, Z., Patt-Shamir, B. and Pettie, S. (2008) Improved Distributed Approximate Matching. *Proceedings of SPAA 08*, Munich, 14-16 June, pp. 129-136. ACM, New York.
- [41] Nieberg, T. (2008) A Local Approximation Algorithm for Maximum Weight Matching. *Proceedings of CTW 2008*, St. Croix, 11-14 May, pp. 44-47. IEEE, Washington.
- [42] Ganeriwal S., Kumar, R. and Srivastava, M. B. (2003) Timing-sync Protocol for Sensor Networks. *Proceedings of SENSYS 2003*, Los Angeles, 5-7 November, pp. 138-149, ACM, New York.
- [43] Rao, S. (2008) *Distributed Systems: An Algorithmic Approach*. CRC Press.
- [44] Karl, H. and Willig, A. (2007) *Protocols and Architectures for Wireless Sensor Networks*. Wiley-Interscience, Malden.
- [45] Woo, A., Tong, T. and Culler, D. (2003) Taming the underlying Challenges of Reliable Multihop Routing in Sensor Networks. *Proceedings of SENSYS 03*, Los Angeles, 5-7 November, pp. 14-27, ACM, New York.
- [46] Dagdeviren, O. and Erciyes, K. (2006) A Distributed Backbone Formation Algorithm for Mobile Ad hoc Networks. *ISPA 2006*, Sorrento, 4-6 December, LNCS 4330, pp. 219-230. Springer-Verlag, Berlin.

10. APPENDIX

In this section, the detailed pseudo-codes of MASC and MCUBA are given. The main algorithm and the helper procedures of MASC are shown in Alg. 1 and Alg. 2. MCUBA's pseudo-codes and the other needed procedures are shown in Alg. 3, Alg. 4, Alg. 5, and Alg. 6.

Algorithm 1 MASC for node_{*j*} receiving message from node_{*i*}

```

1: initially cur_state_j: the current state of nodej
2:   cur_state ← LEADER
3:   round ← 0
4:   msg: the received message, includes: Ci, RSSi,
   leaderi
5:   Ci: the set of nodes in the same cluster with nodei
6:   RSSi: the second largest RSS value of a cluster
   head nodei, used to determine the leadership
7:    $\Gamma_j$ : the set of neighbors of nodej which do not
   belong to the cluster of nodej, given as the parameter
8:   Mw(A): the operator which returns the id of the
   node in set A which has the largest weight of the edge
   connecting to nodej
9:   candidatej ← Mw( $\Gamma_j$ )
10:  m_rounds: number of maximum rounds, given as
   the parameter
11:  R: the set of collected requests
12:  Legend :  $\square$  State  $\wedge$  input_message  $\longrightarrow$  actions
13: procedure MAIN( $\Gamma_j$ , m_rounds)
14:   loop
15:      $\square$  LEADER  $\wedge$  Request  $\longrightarrow$ 
16:       call RQST( msg, m_rounds,  $\Gamma_j$ )
17:      $\square$  LEADER  $\wedge$  New_Round_Interrupt  $\longrightarrow$ 
18:       R ←  $\emptyset$ 
19:       candidatej ← Mw( $\Gamma_j$ )
20:       send REQUEST to candidatej
21:       cur_statej ← CANDIDATE_WT_ACK
22:      $\square$  LEADER  $\wedge$  Drop  $\longrightarrow$ 
23:       call DRP (msg, m_rounds,  $\Gamma_j$ )
24:      $\square$  MATCHED_LEADER  $\wedge$  Request  $\longrightarrow$ 
25:       send DROP to nodei
26:      $\square$  MATCHED_LEADER  $\wedge$  End_Of_Round  $\longrightarrow$ 
27:       cur_statej ← LEADER
28:      $\square$  CANDIDATE_WT_ACK  $\wedge$  Request  $\longrightarrow$ 
29:       call RQST(msg, m_rounds,  $\Gamma_j$ )
30:      $\square$  CANDIDATE_WT_ACK  $\wedge$  Request_Ack  $\longrightarrow$ 
31:       call CLSTR (msg, m_rounds,  $\Gamma_j$ )
32:      $\square$  CANDIDATE_WT_ACK  $\wedge$  Drop  $\longrightarrow$ 
33:       call DRP (msg, m_rounds,  $\Gamma_j$ )
34:      $\square$  MEMBER  $\wedge$  Request  $\longrightarrow$ 
35:       if idi  $\in$  Cj then
36:         send Request_Ack
37:       else forward Request to leaderj
38:       end if
39:   end loop
40: end procedure

```

Algorithm 2 MASC for node_j receiving message from node_i (part 2)

```

41: procedure RQST(msg, m_rounds,  $\Gamma_j$ )
42:   if leaderi = idj then
43:     send Request_Ack
44:   else if idi = candidatej then
45:     call CLSTR(msg, m_rounds,  $\Gamma_j$ )
46:   else
47:      $R \leftarrow R \cup id_i$ 
48:   end if
49: end procedure
50: procedure CLSTR(msg, m_rounds,  $\Gamma_j$ )
51:   send Request_Ack
52:   if ( $RSS_i > RSS_j$ ) or ( $RSS_i = RSS_j$  and  $id_i > id_j$ )
then
53:     cur_statej  $\leftarrow$  MEMBER
54:     leaderj  $\leftarrow$  idi
55:   else
56:     if round < m_rounds then
57:       round  $\leftarrow$  round + 1
58:       set Round_Timer
59:     end if
60:     cur_statej  $\leftarrow$  MATCHED_LEADER
61:      $\Gamma_j \leftarrow \Gamma_j \cup \Gamma_i$ 
62:   end if
63: end procedure
64: procedure DRP(msg, m_rounds,  $\Gamma_j$ )
65:    $\Gamma_j \leftarrow \Gamma_j / id_i$ 
66:   candidatej  $\leftarrow$   $M_w(\Gamma_j)$ 
67:   if candidatej =  $\emptyset$  then
68:     if round < m_rounds then
69:       round  $\leftarrow$  round + 1
70:       set Round_Timer
71:     end if
72:     cur_statej  $\leftarrow$  MATCHED_LEADER
73:   else
74:     if candidatej  $\in$  R then
75:       call CLSTR(msg, candidatej, m_rounds,  $\Gamma_j$ )
76:     end if
77:   end if
78: end procedure

```

Algorithm 3 MCUBA for node_j receiving message from node_i

```

1: initially cur_statej: the current state of nodej
2:   cur_state  $\leftarrow$  LEADER
3: c_level: the cluster level
4:   c_level  $\leftarrow$  0
5: uc_level: the upper cluster level
6: b_candidate: the backbone candidate
7:   msg: is the received message, includes:  $C_i$ ,  $RSS_i$ ,
   leaderi
8:    $C_i$ : the set of nodes in the same cluster with nodei
9:    $RSS_i$ : the second largest  $RSS$  value of a cluster
   head nodei, used to determine the leadership.
10:   $\Gamma_j$ : the set of neighbors of nodej which do not
   belong to the cluster of nodej
11:   $\Gamma_b$ : the set of neighbors of sink node which do not
   belong to the backbone
12:   $M_w(A)$ : the operator which returns the id of the
   node in set A which has the largest weight of the edge
   connecting to nodej.

```

Algorithm 4 MCUBA for node_j receiving message from node_i (Part 2)

```

13:   R: the set of collected Request msgs.
14:   BR: the received Backbone_Request msg.
15:   D: the set of collected Drop msgs.
16:   Legend :  $\square$  State  $\wedge$  input_msg  $\rightarrow$  actions
17: procedure MAIN( $\Gamma_j$ , uc_level)
18:   loop
19:      $\square$  LEADER  $\wedge$  Request  $\rightarrow$ 
20:       call RQST(msg, uc_level,  $\Gamma_j$ )
21:      $\square$  LEADER  $\wedge$  Ldr_TOUT  $\rightarrow$ 
22:       call LDR(msg, uc_level,  $\Gamma_j$ )
23:      $\square$  LEADER  $\wedge$  Drop  $\rightarrow$ 
24:       call DRP(msg, uc_level,  $\Gamma_j$ )
25:      $\square$  LEADER  $\wedge$  Backbone_Request  $\rightarrow$ 
26:       BR  $\leftarrow$  msg
27:      $\square$  LEADER_END  $\wedge$  Request  $\rightarrow$ 
28:       call RQST_END(msg)
29:      $\square$  LEADER_END  $\wedge$  Backbone_Request  $\rightarrow$ 
30:       send Backbone_Request_Ack to nodei
31:      $\square$  CANDIDATE_WT_ACK  $\wedge$  Request  $\rightarrow$ 
32:       call RQST(msg, uc_level,  $\Gamma_j$ )
33:      $\square$  CANDIDATE_WT_ACK  $\wedge$  Drop  $\rightarrow$ 
34:       call DRP(msg, uc_level,  $\Gamma_j$ )
35:      $\square$  CANDIDATE_WT_ACK  $\wedge$  Request_Ack  $\rightarrow$ 
36:       call CLSTR(msg, uc_level,  $\Gamma_j$ )
37:      $\square$  CANDIDATE_WT_ACK  $\wedge$  Backbone_Request
38:        $\rightarrow$  BR  $\leftarrow$  msg
39:      $\square$  MEMBER  $\wedge$  Request  $\rightarrow$ 
40:       if  $id_i \in C_j$  then
41:         send Request_Ack to nodei
42:       else
43:         forward Request to leaderj
44:       end if
45:      $\square$  MEMBER  $\wedge$  Backbone_Request  $\rightarrow$ 
46:       forward Backbone_Request to nodej
47:      $\square$  MEMBER_END  $\wedge$  Request  $\rightarrow$ 
48:       call RQST_END(msg)
49:      $\square$  MEMBER_END  $\wedge$  Backbone_Request  $\rightarrow$ 
50:       forward Backbone_Request to leaderj
51:      $\square$  SINK_WT_FOR_BACKBONE  $\wedge$  Request  $\rightarrow$ 
52:       send Drop to nodei
53:      $\square$  SINK_WT_FOR_BACKBONE  $\wedge$ 
54:       Backbone_Request_Ack  $\rightarrow$ 
55:       call BCKBN(msg,  $\Gamma_j$ )
56:   end loop
57: end procedure
58: procedure RQST(msg, uc_level,  $\Gamma_j$ )
59:   if leaderi = idj then
60:     send Request_Ack
61:   else if idi = candidatej then
62:     call CLSTR(msg, uc_level,  $\Gamma_j$ )
63:   else
64:      $R \leftarrow R \cup id_i$ 
65:   end if
66: end procedure

```

Algorithm 5 MCUBA for node_j receiving message from node_i (Part 3)

```

67: procedure CLSTR(msg, uc_level,  $\Gamma_j$ )
68:   send Request_Ack
69:   leader_j  $\leftarrow id_i$ 
70:   cluster_level_j  $\leftarrow cluster\_level_j + cluster\_level_i$ 
71:   if ( $id_i = id_{sink}$ ) or ( $RSS_i > RSS_j$ ) or ( $RSS_i =$ 
    RSS_j) and  $id_i > id_j$ ) then
72:     forward all REQUESTS in R to leader_j
73:     forward BR to leader_j
74:     if cluster_level_j  $\geq uc\_level$  then
75:       cur_state_j  $\leftarrow MEMBER\_END$ 
76:     else
77:       cur_state_j  $\leftarrow MEMBER$ 
78:     end if
79:   else
80:      $\Gamma_j \leftarrow \Gamma_j \cup \Gamma_i$ 
81:     candidate_j  $\leftarrow M_w(\Gamma_j)$ 
82:     if cluster_level_j  $\geq uc\_level$  and candidate_j  $\neq$ 
     $\emptyset$  then
83:       cur_state_j  $\leftarrow CANDIDATE\_WT\_ACK$ 
84:     else
85:       send Drop to all node  $\in R$ 
86:       R  $\leftarrow \emptyset$ 
87:       if  $id_j = sink_j$  then
88:          $\Gamma_b \leftarrow \Gamma_j$ 
89:         b_candidate_j  $\leftarrow M_w(\Gamma_b)$ 
90:         cur_state_j  $\leftarrow$ 
91:         SINK\_WT\_FOR\_BACKBONE
92:         if b_candidate_j  $\neq \emptyset$  then
93:           send Backbone_Request to
94:           b_candidate
95:         else
96:           cur_state_j  $\leftarrow SINK\_END$ 
97:         end if
98:       else
99:         cur_state_j  $\leftarrow LEADER\_END$ 
100:        if BR  $\neq \emptyset$  then
101:          send Backbone_Request_Ack to BR_i
102:        end if
103:      end if
104:    end if
105:  end if
106: end procedure
107: procedure RQST_END(msg)
108:   if  $id_i \in C_j$  then
109:     send Request_Ack to nodei
110:   else
111:     send Drop to nodei
112:   end if
113: end procedure
114: procedure BCKBN(msg,  $\Gamma_j$ )
115:    $\Gamma_b \leftarrow \Gamma_b \cup \Gamma_i$ 
116:   b_candidate_j  $\leftarrow M_w(\Gamma_b)$ 
117:   if b_candidate_j  $\neq \emptyset$  then
118:     send Backbone_Request to b_candidate_j
119:   else
120:     b_candidate_j  $\leftarrow SINK\_END$ 
121:   end if
122: end procedure

```

Algorithm 6 MCUBA for node_j receiving message from node_i (Part 4)

```

123: procedure DROP(msg, uc_level,  $\Gamma_j$ )
124:    $\Gamma_j \leftarrow \Gamma_j / id_i$ 
125:   candidate_j  $\leftarrow M_w(\Gamma_j)$ 
126:   if candidate_j =  $\emptyset$  then
127:     cur_state  $\leftarrow MATCHED\_LEADER$ 
128:   else
129:     if candidate_j  $\in R$  then
130:       call CLSTR(msgcandidate_j, uc_level,  $\Gamma_j$ )
131:     end if
132:   end if
133: end procedure
134: procedure LDR(msg, uc_level,  $\Gamma_j$ )
135:   if cluster_level  $< uc\_level$  then
136:     Request to candidate_j
137:     cur_state  $\leftarrow CANDIDATE\_WT\_ACK$ 
138:   else
139:     send Drop to all node  $\in R$ 
140:     if  $id_j \neq id_{sink}$  then
141:       cur_state_j  $\leftarrow LEADER\_END$ 
142:       if BR  $\neq \emptyset$  then
143:         send Backbone_Request_Ack to BR_i
144:       end if
145:     else
146:        $\Gamma_b \leftarrow \Gamma_j$ 
147:       b_candidate_j  $\leftarrow M_w(\Gamma_b)$ 
148:       cur_state_j  $\leftarrow$ 
149:       SINK\_WT\_FOR\_BACKBONE
150:       if b_candidate_j  $\neq \emptyset$  then
151:         send BACKBONE\_REQUEST to
152:         b_candidate
153:       else
154:         cur_state_j  $\leftarrow SINK\_END$ 
155:       end if
156:     end if
157:   end procedure

```
