Advance Access publication on February 7, 2010
doi:10.1093/comjnl/bxq004

Graph Matching-Based Distributed Clustering and Backbone Formation Algorithms for Sensor Networks

ORHAN DAGDEVIREN[1,*] AND KAYHAN ERCIYES[2]

[1] *Computer Engineering Department, Izmir Institute of Technology, Urla, Izmir 35430, Turkey*
[2] *Computer Engineering Department, Izmir University, Gursel Aksel Bulv., No. 14, Uckuyular, Izmir 35350, Turkey*
**Corresponding author: orhandagdeviren@iyte.edu.tr; orhandagdeviren@yahoo.com*

Clustering is a widely used technique to manage the essential operations such as routing and data aggregation in wireless sensor networks (WSNs). We propose two new graph-theoretic distributed clustering algorithms for WSNs that use a 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 an 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 a 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.

Keywords: sensor networks, clustering, backbone formation, graph matching

Received 13 August 2009; revised 28 November 2009
Handling editor: Yu-Chee Tseng

1. INTRODUCTION

Wireless sensor networks (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 tasks 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 the literature for clustering in WSNs. LEACH [1] assumes that cluster heads consume uniform energy and it provides random rotation of cluster heads. Cluster heads in HEED [2] are elected using a probabilistic method that 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.

In [6], we find one of the algorithms that performs graph-theoretic clustering in sensor networks. Examples of graph-theoretic clustering algorithms for various networks are given in [7–14]. 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, the 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, then 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 the 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 the total number of messages. To the best of our knowledge, these two algorithms are the first algorithms in the 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 Sections 4 and 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 Appendix.

2. BACKGROUND

2.1. Network model

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

(i) Each node has a distinct *node_id*.
(ii) The nodes are stationary.
(iii) Links between nodes are symmetric. Thus if there is a link from u to v, there exists a reverse link from v to u.
(iv) Nodes do not know their positions. They are not equipped with a position tracker like a GPS receiver.
(v) All nodes are equal in terms of processing capabilities, radio, battery and memory.
(vi) Each node knows its neighbors and the quality of the links connected to the neighbors. The quality of a link is the same for both nodes attached to it.

On the basis of these assumptions, the network may be modeled as a weighted undirected graph $G_w(V, E_w)$, where V is the set of vertices, E_w is the set of the edges with weights. The weights of the edges can be represented by the link quality indicators supported by various underlying protocols as in [18, 19]. An example weighted undirected graph model is depicted in Fig. 1 where the transmission ranges of the nodes are shown with dotted circles, and the weight of each edge corresponds to its link quality.

The other popular model used in graph-theoretical approaches is the unit disk graph model where each node is

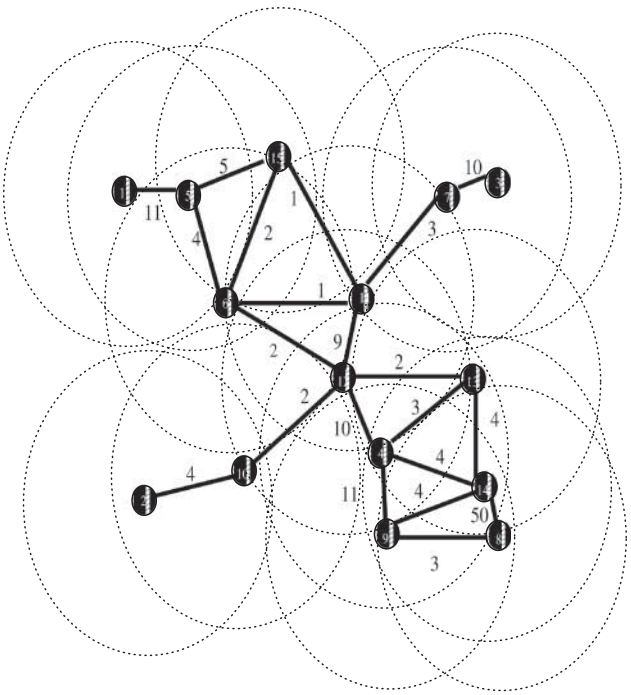


FIGURE 1. Weighted undirected graph model.

identified with a disk of unit radius r=1, and there is an edge between two nodes u and v if and only if the distance between u and v is at most 1 [20, 21]. Unit disk graphs captures the behavior of broadcast radio transmission; thus it is good for modeling *ad hoc* and sensor networks [20], but this model lacks representation of link qualities. A weighted undirected graph model not only models the radio transmission but also represents the link qualities by the edge weights.

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 a star topology in which the cluster head is at the center of the star, whereas 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 its cluster is overcrowded. On the other hand, construction of clusters with very small sizes may not be cost effective. In conclusion, the 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 with various MAC and physical layer standards such as in [18, 19, 23–26].
(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 (DS) based and spanning tree based. A DS 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 DS is connected, then the DS is called a 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 and Wu [10], Nanuvala [11], and Cokuslu *et al.* [12] added extra heuristics to Wu's algorithm to reduce the size of the CDS Yan *et al.* [13] proposed an energy-efficient DS-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 *et al.* [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 a *PARENT*(*nhops*) message to its neighbors to reinitiate the operation. Each node sends the *PARENT*((*nhops* + 1)*mod depth*) message to its neighbors upon the first reception of the *PARENT*(*nhops*) message. The recipients 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 the following five 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 the 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' algorithm with 1/2 approximation ratio [34], Preis' algorithm with 1/2 approximation ratio [35], Drake's algorithm with 2/3-ϵ approximation ratio [36], Petite's algorithm with 2/3-ϵ 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' 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 the *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, the 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 the *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 the construction of a maximal independent set which will result in high time and message complexities for our implementation, and 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 Algorithms 1 and 2.

In a clustering session, a node first 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, the destination node will reply with *REQUEST_ACK* and the clustering operation will be completed. Otherwise, if the destination node is matched with another node, it will reply with a *DROP* message and will search for 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:

(i) *LEADER*: *LEADER* is the cluster head of its cluster. All the nodes are in the *LEADER* state at the beginning of the algorithm execution. If a *LEADER* receives a *REQUEST*

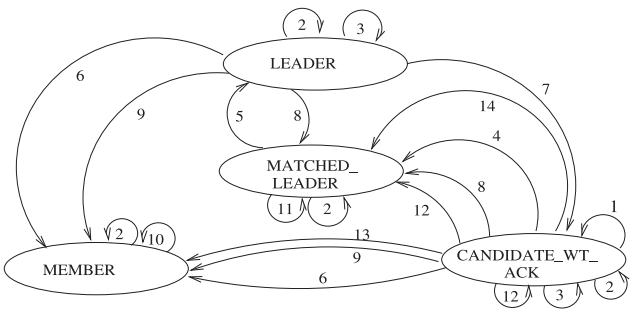


FIGURE 2. FSM of MASC.

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 a *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 Received Signal Strength (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$_i$ and that of node$_j$ is id$_j$.
(2) The node with the largest RSS is the new leader and makes a state transition to the *MATCHED_LEADER* state, the other node becomes 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 the node connected on the edge with the second largest RSS will be the next candidate. The *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.

(ii) *MATCHED_LEADER*: A node in this state rejects the clustering requests by sending a DROP message after which it makes a state transition to the LEADER state.
(iii) *MEMBER*: Cluster member nodes are in the *MEMBER* state and they forward their requests to their cluster head.
(iv) *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 a *CANDIDATE_WT_ACK* state. If a *DROP* message is received, a new candidate is selected. If no other candidate exists, the node will make a transition to a *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 three rounds using MASC is as follows. In the first round, a *NEW_ROUND_INTERRUPT* occurs in node 13. Node 13 is initially in a *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, and so it replies with a *REQUEST_ACK* message. The second largest RSS value of the node 13 is node 5 and node

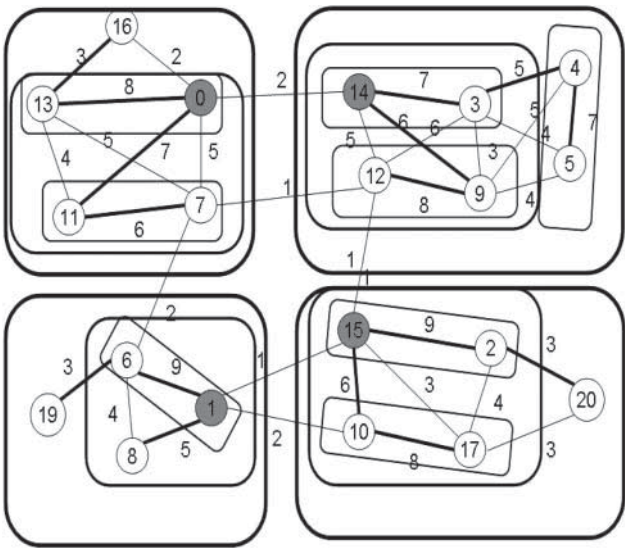


FIGURE 3. An example operation.

0 is node 7 and hence node 0 makes a state transition to the *MATCHED_LEADER* state and node 13 makes a state transition to the *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 make 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 cannot find any cluster to join in the first round.

In the beginning of the second round, node 12 sends a *REQUEST* message to node 3. Node 3 which is in the *MEMBER* state receives the *REQUEST* message and forwards it 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 follows: {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 weighted graph matching as the method to select the strong communication links for clustering, different from the

other graph-theoretic algorithms. Also the number of clusters can be controlled by the number of rounds different from the DS-based clustering algorithms [9–13]. The advantages of MASC over other graph-theoretic approaches is described in Sections 6 and 7. On the other hand, MASC lacks some important design issues as follows:

(i) 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].
(ii) The energy consumption of the algorithm execution is greater than the proposed graph-theoretic approaches as shown in Sections 6 and 7.
(iii) 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 the same weighted matching 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

The MCUBA algorithm constructs clusters and a backbone concurrently and asynchronously. The FSMs for the MCUBA are given in Figs. 4 and 5 and the detailed pseudo-codes are given in Algorithms 3–6. FSM in Fig. 4 is used by all of the nodes except the sink node which employs the FSM shown in Fig. 5. The *LEADER*, *CANDIDATE_WT_ACK* and *MEMBER* states of MCUBA are similar to those states of MASC with small modifications. Initially all the nodes except the sink node executing MCUBA are in a *LEADER* state. After an 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 the *REQUEST_ACK* message, it can make a state transition to either a *LEADER*, *LEADER_END*, *MEMBER* or *MEMBER_END* state. The basic message flow and leader election procedure of MCUBA is similar to that of MASC.

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 the *LEADER_END* state drops all of the clustering requests. The same procedure is applied to the member nodes. A node in the *MEMBER_END* state drops all clustering requests 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, the leader node replies with a *REQUEST_ACK* message. This approach is different from that of MASC. In MASC, a node in the *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 from other nodes, the 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, the sink node selects the locally heaviest edge outgoing from its cluster. When the cluster level of sink node reaches the *upper_level*, the sink node makes a state transition from a *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 has already 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 the sink node receives a *BACKBONE_REQUEST_ACK*, it merges the source cluster and its own cluster, and then continues the same operation described below until all the 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. The overhearing technique is applied to reduce the number of ACK messages to decrease the energy consumption. In Fig. 6, a sensor network with three 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

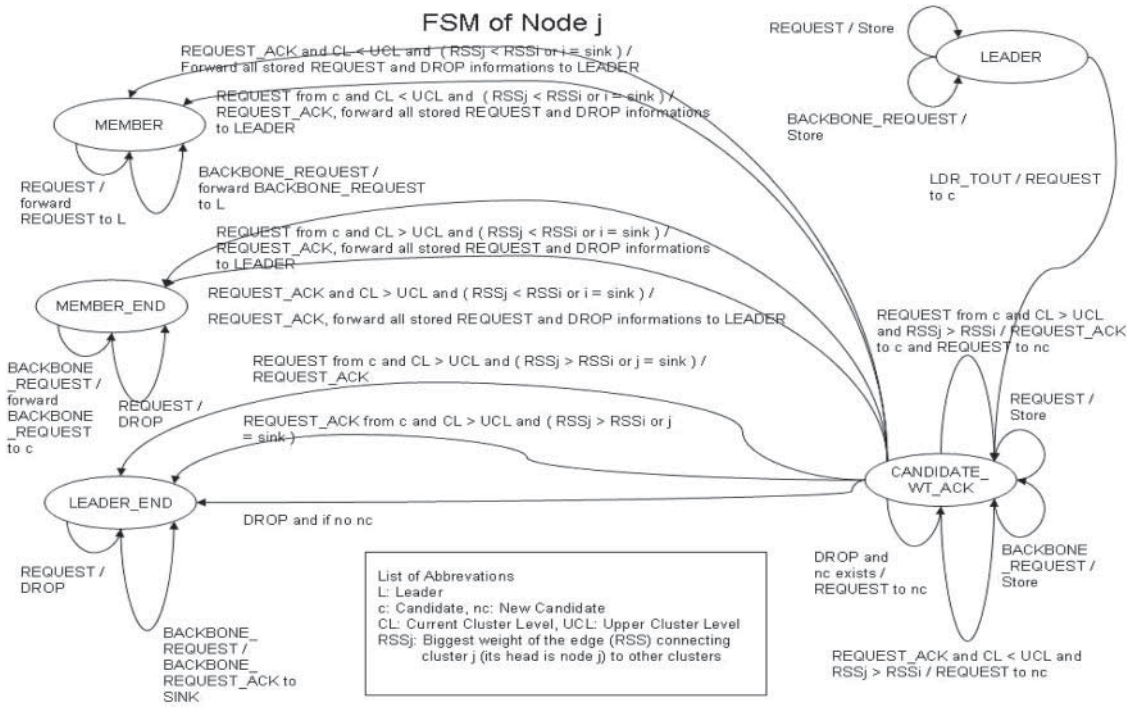
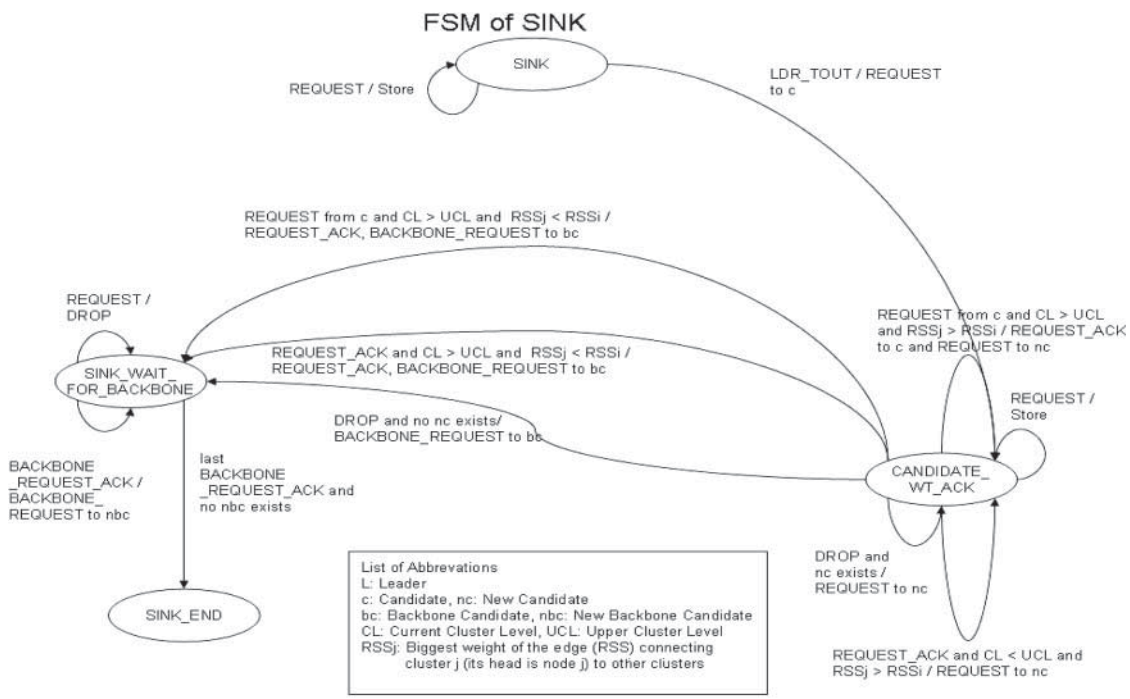


FIGURE 4. MCUBA's FSM for node j.



MCUBA's FSM for sink.

FIGURE 5. MCUBA's FSM for sink.

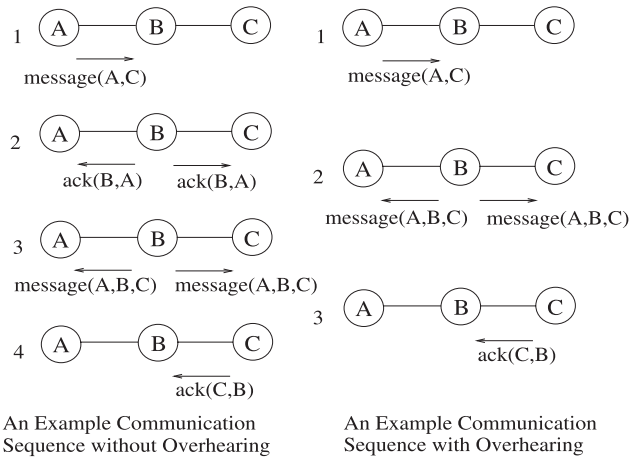


FIGURE 6. Overhearing technique.

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.

5.4. An example operation

An example operation of the 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 a *LEADER* state; the sink node is in a *SINK* state. Node 0's candidate is node 16, since the heaviest edge outgoing from node 0 is the edge that connects node 0 to node 16. Node 0 sends a *REQUEST* message to node 16. Node 16's candidate is node 0, and hence node 16 replies with a *REQUEST_ACK* message to node 0 and they construct a new cluster of 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 and that of node 19 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 operations explained above.

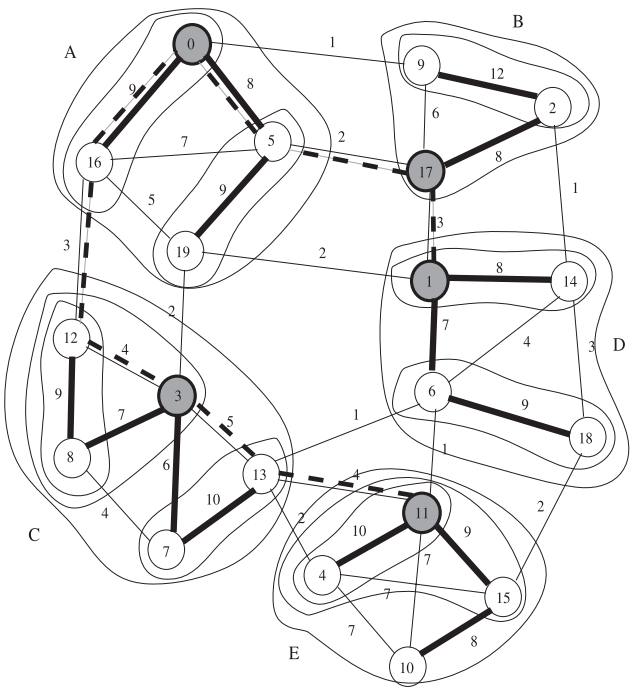


FIGURE 7. An example operation of MCUBA.

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 cluster is formed as: {17, 2, 9}, and the new leader of this cluster is node 17. The other clusters of level 3 are constructed as follows: {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, and hence node 6 sends a *REQUEST_ACK* message to node 1 and cluster D is formed as follows: {1, 14, 6, 18} as shown in Fig. 7. The leader of this cluster, node 1, makes a transition to the *LEADER_END* state since its cluster level reaches the *upper_level*. The nodes in the *LEADER_END* state reject all clustering requests. Node 0 merges its cluster with node 5's cluster and cluster A is formed as follows: {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 the *SINK_WT_FOR_BACKBONE* state, and starts a 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 the *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 its 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 the *LEADER_END* state. The cluster B is formed as follows: {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 follows: {3, 12, 8, 13, 7}, and the cluster D is produced as follows:{3, 12, 8, 13, 7} shown in Fig. 7. Selected edges for communication during the clustering sessions are shown as bold in Fig. 7.

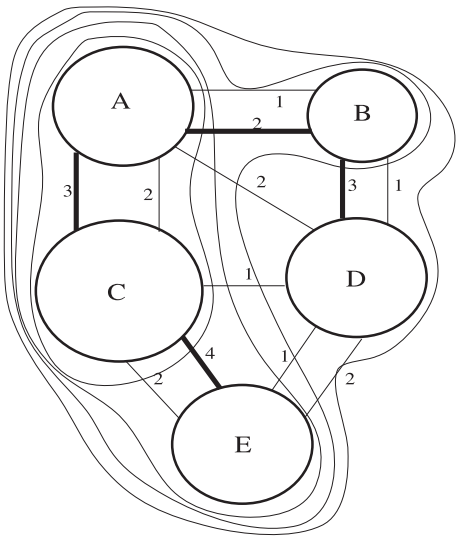


FIGURE 8. An example backbone formation of MCUBA.

The leader of cluster C, node 3, sends a *BACKBONE_REQUEST_ACK* message in reply to the recorded *BACKBONE_REQUEST* after the 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 that connects the backbone to other clusters. Node 11 is in the *LEADER_END* state and replies with a *BACKBONE_REQUEST_ACK* message after it receives the *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 the 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 the MEMBER states are the cluster members, the nodes in the MATCHED_LEADER state are the cluster heads. In the MCUBA, the nodes in the MEMBER and MEMBER_END state are the cluster members, the nodes in the LEADER_END state are the cluster heads and the node in the SINK_END state is the sink node.*

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

LEMMA 6.1. *In MASC and the MCUBA, if a node sends a REQUEST message to its candidate, then 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 a *LEADER* or *CANDIDATE_WT_ACK* state. It sends a *REQUEST_ACK* message.
Case 2: The candidate is in a *MATCHED_LEADER* state. It sends a *DROP* message.
Case 3: The candidate is in a *MEMBER* state. It forwards the message to its leader. Due to Observation 2, the leader cannot be in the *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 the MEMBER state and no state transitions before the end of the round is possible for a node in the MATCHED_LEADER state. In MCUBA, no state transitions are possible for a node in a MEMBER, MEMBER_END, LEADER_END or SINK_END state.*

Proof. In MASC, a node makes a state transition from a *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 Figs. 2,4 and 5, with the pseudo-codes in Algorithms 1–6. □

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 the following cases.

Case 1: The node is in the *LEADER_END* state. In this state, the node replies with a *BACKBONE_REQUEST_ACK* message.
Case 2: The node is in the *MEMBER* or *MEMBER_END* state. The node forwards the *BACKBONE_REQUEST* message to its leader.
Case 3: The node is in the *LEADER* or *CANDIDATE_WT_ACK* state. It records the *BACKBONE_REQUEST* message. If the node makes a state transition to the *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 the *LEADER_END* state, it sends a *BACKBONE_REQUEST_ACK* message.
Case 4: The node is in the *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 Figs. 4 and 5, with the pseudo-codes in Algorithms 3–6, 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 the *MEMBER* or *MATCHED_LEADER* state. We assume to the contrary that a node is in the *CANDIDATE_WT_ACK* or *LEADER* state after the execution of the algorithm. We investigate each of the following cases:

Case 1: The node is in the *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, the node makes a state transition. If the nodes does not receive any *REQUEST* message, a *LEADER_TOUT* occurs, it makes a state transition to a *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 the *MEMBER* or *MATCHED_LEADER* state. If the node does not have any candidate, a *LEADER_TOUT* occurs and it makes a state transition to the *MATCHED_LEADER* state.
Case 2: The node is in a *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 the *MATCHED_LEADER* or *MEMBER* state. If the node receives a *DROP* message, it makes a state transition to the *MATCHED_LEADER* state or sends a *REQUEST* message to its candidate. According to Lemma 6.1, if a node sends a *REQUEST* message to its candidate, it receives a *REQUEST_ACK* or *DROP* message.

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

THEOREM 6.2. *After the execution of the 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 the *LEADER*, *CANDIDATE_WT_ACK*, *SINK* or *SINK_WT_FOR_BACKBONE* state. Each case is investigated below:

Case 1: The node is in the *LEADER* state. The node must not receive any message from its candidate or send any message 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 the *LEADER* state. If the node has no candidate, it makes a state transition to the *LEADER_END* state.
Case 2: The node is in the *SINK* state. The possibilities are the same as in case (1). The node makes a state transition to the *CANDIDATE_WT_ACK*, *SINK_WT_FOR_BACKBONE* or *SINK _E N D* state.
Case 3: The node is in the *CANDIDATE_WT_ACK* state. The node must not receive any message. By Lemma 6.1, node receives a *DROP* or *REQUEST_ACK* message.
Case 4: The node is in the *SINK_WT_FOR_BACKBONE* state. The node must not receive a *BACKBONE_REQUEST_ACK* message for each *BACKBONE_REQUEST* message; but by Lemma 6.3, each *BACKBONE_REQUEST* message is given a reply.

In all cases, each node makes a state transition to the *MEMBER*, *MEMBER_END*, *LEADER_END* or *SINK_END* state. By 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. By Observation 1, in a clustering session of MCUBA and MASC, the source node sends a *REQUEST* message, and the destination node replies with a *DROP* or *REQUEST_ACK* message. There must be a problem in this message flow. Nevertheless, it is proved in Lemma 6.1, Lemma 6.2 and Theorem 6.1, that all message flows related to clustering are successful. By 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. Nevertheless, it is proved in Lemma 6.3 and Theorem 6.2 that all message flows related to backbone formation are successful. We contradict with our assumption. □

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: If $r = 1$, then the algorithm runs same with Hoepman's distributed matching. Thus the upper bound is 2.

Induction step: We assume that after the rth 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 the $(r+1)$th round, the upper bound is $2^r+2^r=2^{r+1}$. □

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

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

6.3. Selected edge weights

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

(i) W: The total selected edge weight for the clustering operation by using perfect weighted matching.
(ii) B: The total selected edge weight for the backbone formation operation by using perfect weighted matching.
(iii) A: The total selected edge weight by MASC with r rounds.
(iv) S: The total selected edge weight for the clustering operation by the MCUBA with *upper_level* $= 2^r$ and s = *upper_level*.
(v) C: The total selected edge weight for the backbone formation operation by the 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: If $r = 1$, then the algorithm turns to Hoepman's distributed matching. Thus $A \geq W(1/2)$ is true.

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

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

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

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

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

□

THEOREM 6.6. *The value $S + C$ lies in between as follows:*

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

Proof. We prove by induction.

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

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

For $0 \leq C \leq B$, we have

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

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

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

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

For $n \geq 0$, we have

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

then

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

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 of the 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

$$\frac{2(\Delta-1) + \cdots + 2(2^{r-1}(\Delta-1) + (\Delta-1))}{2^{r-1}}$$
$$= \frac{2(\Delta-1)(1 + 2 + \cdots + (2^{r-1}+1))}{2^{r-1}}$$
$$= \frac{(\Delta-1)(2^{r-1}+1)(2^{r-2}+1)}{2^{r-1}}$$
$$= \left(\frac{\Delta 2^r}{4} + \Delta + \frac{\Delta}{2} + \frac{2\Delta}{2^r} - 2^{r-2} - \frac{3}{2} - \frac{2}{2^r} \right) \in O(\Delta 2^r).$$

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 and has an upper bound of* $O(\Delta s)$.

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

$$\begin{aligned}
&\frac{2(\Delta-1)+\cdots+2(2s-2(\Delta-1)+(\Delta-1))}{2s-2}+\frac{n}{sn}\\
&=\frac{2(\Delta-1)(1+2+\cdots+2s-1)}{2s-2}+\frac{1}{s}\\
&=\frac{(\Delta-1)(2s-1)(2s)}{(2s-2)}+\frac{1}{s}\\
&=\left(\frac{\Delta 2s^2-\Delta 2s-4s^2+2s}{2s-2}+\frac{1}{s}\right)\in O(\Delta s).
\end{aligned}$$

The best case occurs in the complete graph K_n in Fig. 10. Here the n nodes exchange a *REQUEST* and a *REPLY* message. After that the, $n/2$ nodes exchange a *REQUEST* and a *REPLY* message. The operation is as follows. The equation is given by

$$\frac{n(1+1/2+1/4+\cdots)}{n}\leq 2. \qquad \square$$

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

Proof. We consider clustering and backbone formation together. The best case occurs when clusters of level 1 are connected to form clusters of level 2, clusters of level 2 are connected to form clusters of level 4, and the operation continues similarly. Figure 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 $O(n)$. Figure 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 (energy consumptions other than radio are negligible). Overhearing reduces at least* 40% *of the energy consumed at an average clustering session of the MCUBA with* $upper\_level \geq 8$.

Proof. Let s be the *upper_level*. Since the 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 $s/2$. Figure 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 the MCUBA, a *CONTROL_ACK* message should be sent so that the total message transfers for an average clustering session

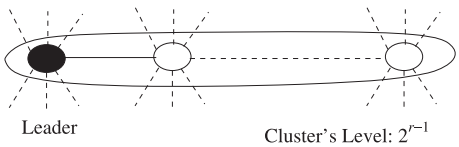


FIGURE 9. Topology for worst case message complexity of MASC.

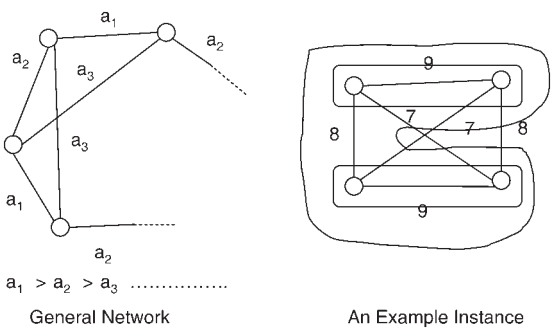


FIGURE 10. Topology for best case message complexity of MCUBA.

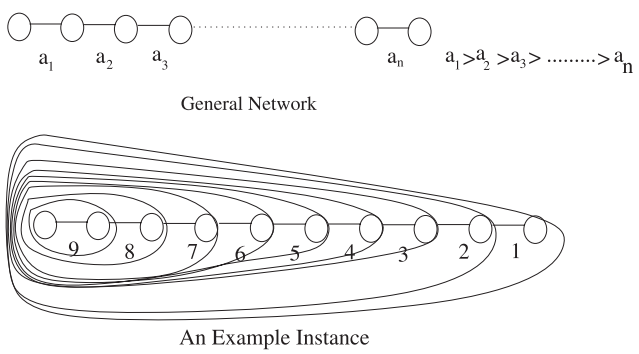


FIGURE 11. Topology for worst case time complexity of MCUBA.

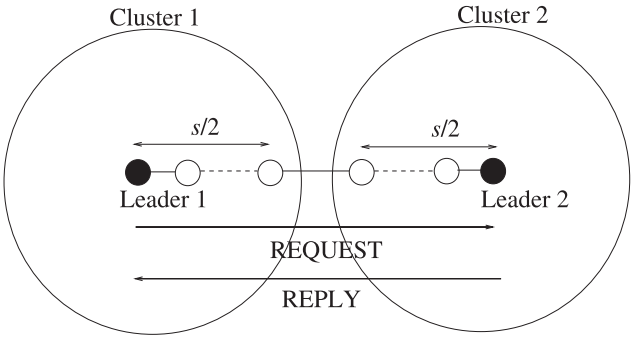


FIGURE 12. An average clustering session.

without overhearing is (N):

$$\begin{aligned} N &= (\textit{REQUEST} \text{ messages} + \textit{REPLY} \text{ messages} \\ &\quad + \textit{CONTROL\_ACK} \text{ messages}) \\ &= ((s/2+1) + (s/2+1) + (s+2)) \\ &= 2s + 4. \end{aligned}$$

By using overhearing, there is no need for *CONTROL_ACK* messages except the two last messages which are not routed, such that the total number of messages with the overhearing technique is given by

$$\begin{aligned} T &= ((s/2+1) + (s/2+1) + 2) \\ &= s + 4, \end{aligned}$$

and

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

For s $\geq$ 8, we have

$$\frac{s}{2s+4} 100 \geq 40\%. \qquad \square$$

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 the node id 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 the DSTA and a DS-based algorithm. The DSTA constructs clusters and ensures a spanning tree formation rooted at the sink as described in Section 3.1. We set the *depth* parameter of the DSTA to 3 in our experiments. The DS is a CDS-based clustering algorithm, an algorithm similar to Wu's algorithm. Initially all nodes are white in the DS algorithm. The DS has two rules to find the the CDS:

(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, the 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 the *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 the 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 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. Figure 13 shows the number of messages used in Hoepman's algorithm.

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

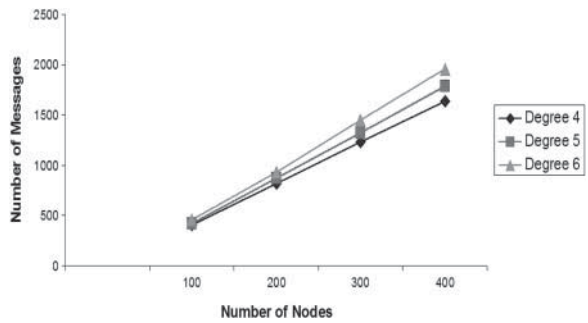


FIGURE 13. Number of messages used in Hoepman's algorithm.

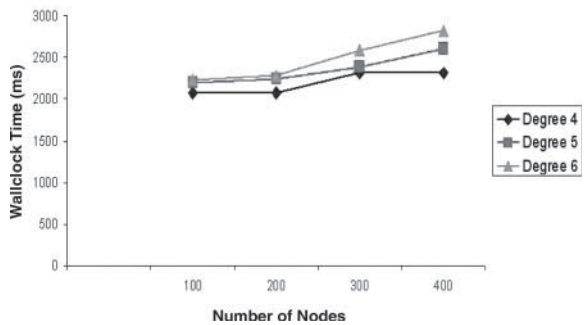


FIGURE 14. Wallclock time values of Hoepman's algorithm.

The 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 2 s to 3 s 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 regard 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 criterion for our algorithms. We measured the total selected edge weights of MASC and the MCUBA. The edge weights were represented with the RSS values defined in IEEE 802.11 [19]. The weight of selected edges in the rth round are heavier than the $(r+1)$th round, and 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 the 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 the 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 follows: 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 the 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

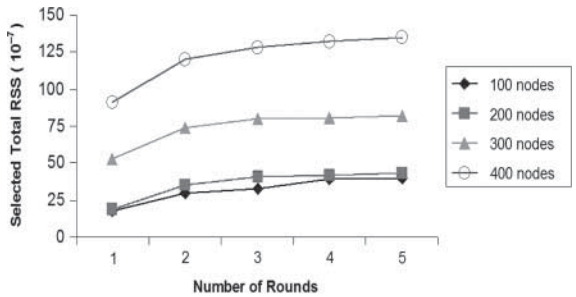


FIGURE 15. Total selected edge weights by MASC.

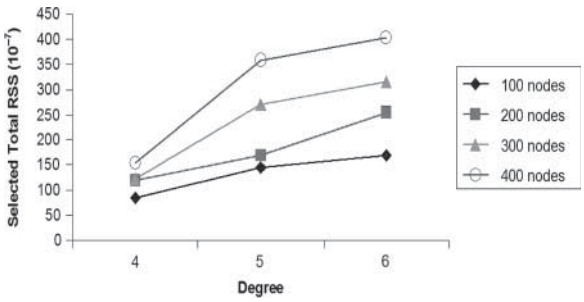


FIGURE 16. Total selected edge weights by MCUBA.

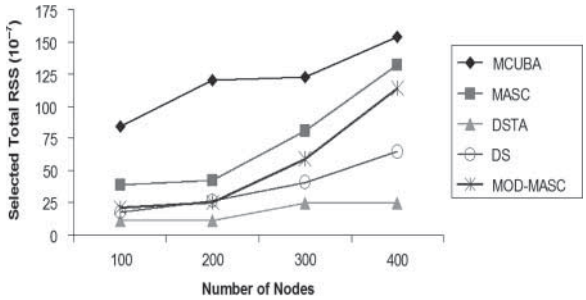


FIGURE 17. Total selected edge weights by algorithms.

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

7.3. Clustering quality

To evaluate the quality of the produced clusters, we used two metrics: the 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 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 half of the number of nodes in the first round and cluster number decreases as the number of rounds increases. In the 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 the MCUBA with *upper_level* $= 10$ is measured against the number of nodes and the average node degree as shown in Fig. 19. The number of clusters produced by the MCUBA is stable and controllable as seen in this figure. We also vary the *upper_level* of the MCUBA between 5, 7, 10 and 20 to measure the effect of the *upper_level* parameter as shown in Fig. 20. The number of produced clusters by the 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. The DS is restricted by the CDS construction and thus number of clusters may not be

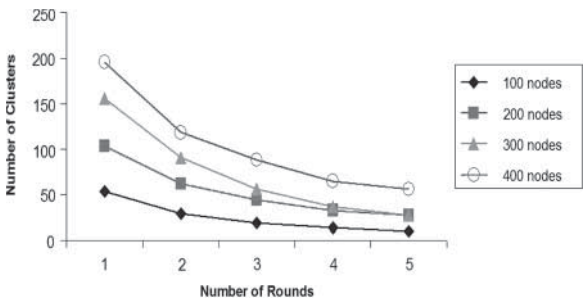


FIGURE 18. Number of clusters produced by MASC.

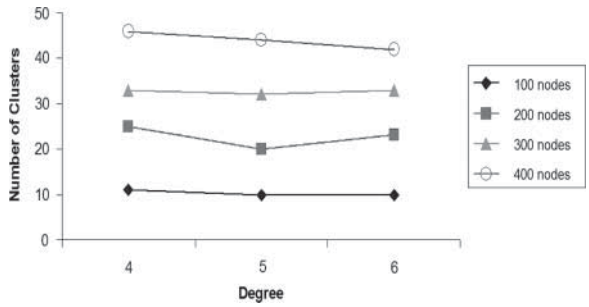


FIGURE 19. Number of clusters produced by MCUBA.

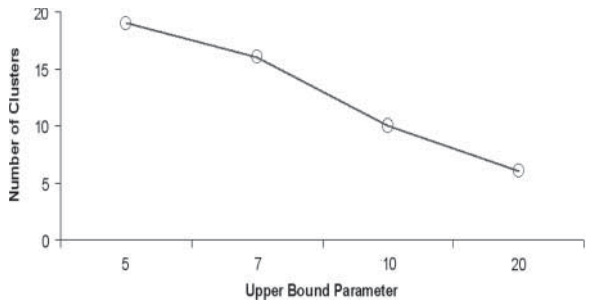


FIGURE 20. Clusters produced by MCUBA against *upper_level*.

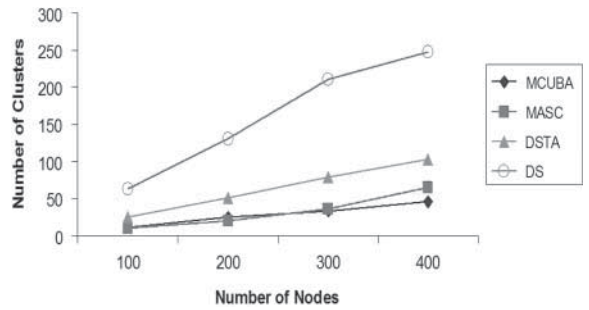


FIGURE 21. Number of clusters produced by algorithms.

controllable. On the other hand, the cluster levels are adjustable in the DSTA with the *depth* parameter. Among all of the implemented algorithms, the DS has the worst performance as shown in Fig. 21 because the number of clusters is more than half of the nodes. The DSTA with *depth* = 3 performs well since the number of clusters increases linearly with a small slope as the number of nodes increases. The MCUBA with *upper_level* = 10 produces 46 clusters, MASC with four rounds produces 65 clusters, the DSTA produces 102 clusters, and the DS produces 247 clusters for the sensor network with 400 nodes having an average degree of 4. The 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, and thus the number of clusters are controllable.

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

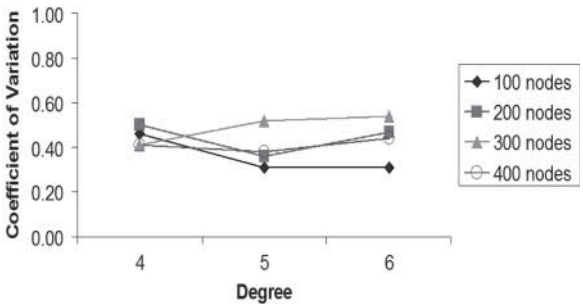


FIGURE 22. CV of clusters produced by MCUBA.

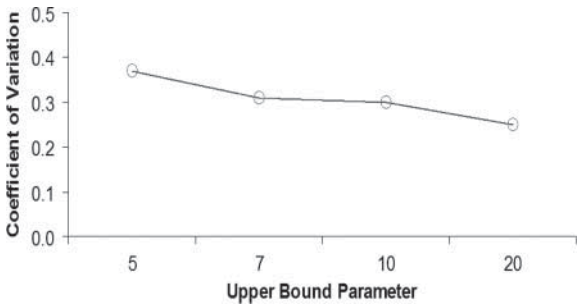


FIGURE 23. CV of clusters produced by MCUBA against *upper_level*.

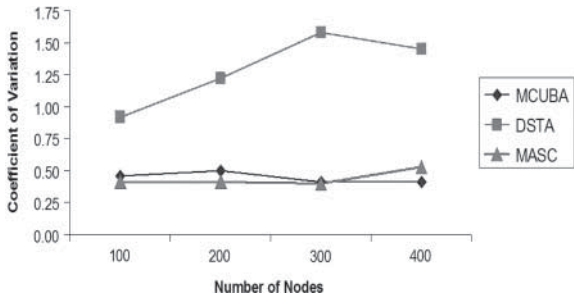


FIGURE 24. CV of clusters produced by algorithms.

measured CV values are stable and vary between 0.40 and 0.25 as seen in Fig. 23. The comparison of the CV values measured by MASC with four rounds, the MCUBA with *upper_level* = 10 and the DSTA with *depth* = 3 is seen in Fig. 24. Since the number of clusters produced by the DS is not controllable, we do not include the DS in this comparison. The average CV values of MASC, the MCUBA and the DSTA are 0.44, 0.45 and 1.29, respectively.

7.4. Energy consumptions

Energy efficiency is an important objective for the 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. In addition, as shown in Fig. 26, the energy consumed by the MCUBA with *upper_level* = 10 increases linearly with the node degree and the number of nodes. We compared the energy consumption of all approaches as seen in Fig. 27. In the DS and the DSTA, nodes exchange less number of messages than those in MASC and MOD-MASC; thus 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, an overhearing technique was applied in the MCUBA. The effect of the overhearing technique applied in

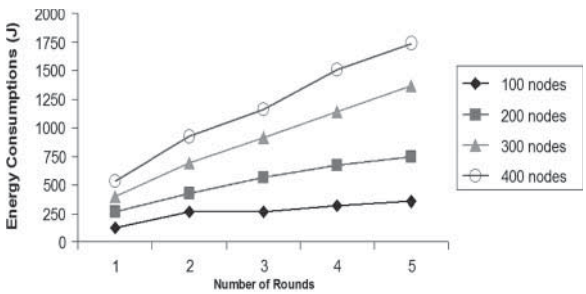


FIGURE 25. Energy consumption of MASC.

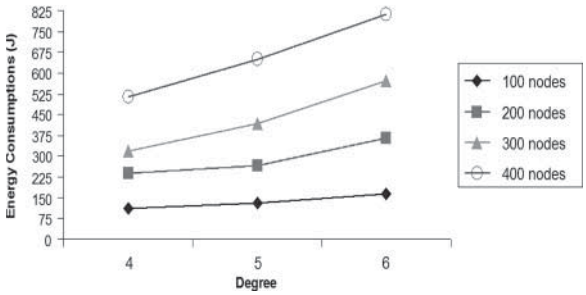


FIGURE 26. Energy consumption of MCUBA.

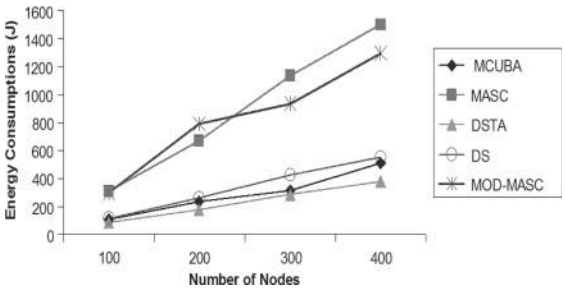


FIGURE 27. Energy consumptions of algorithms.

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

7.5. Wallclock Times

We measured the wallclock times of the MCUBA with *upper_level* = 10, MASC with four rounds, the DSTA with *depth* = 3 and the DS. As seen in Fig. 14, at the worst case, the time for the distributed matching is 3 s approximately. In our measurements, the time for a round of MASC is 5 s in the worst case; thus we set the round of MASC to 5 s. Although the nodes are idle when they finish the execution of the MASC in a round before 5 s, they must wait until the end of the round. Since nodes execute asynchronously in the MCUBA, there is no waiting time for all nodes. The wallclock of MASC with four rounds is constant and equals 20 s. In the DSTA, a node forwards each received message to all neighbors. In the 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, the MCUBA performs the best among the other approaches, the DSTA also performs well, whereas MASC performs the worst due to waiting times.

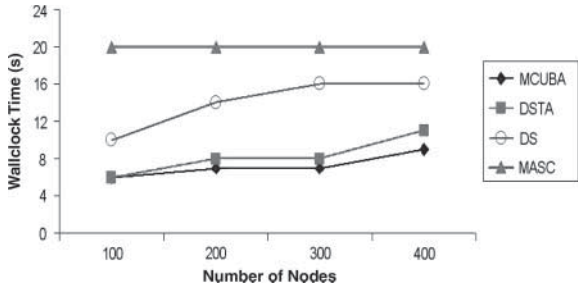


FIGURE 28. Wallclock time of algorithms.

8. DISCUSSIONS

In this section, we discuss the applicability and the extensions of our proposed algorithms. The 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. Quality metrics other than the 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 the MCUBA, DS and DSTA, where each node senses events and sends these 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:

(i) n: number of nodes;
(ii) Δ: average node degree;
(iii) T: the total time for the application;
(iv) p: period time for the clustering algorithms;
(v) E: average energy dissipated in a message transfer;
(vi) V: the total number of events sensed by the nodes;
(vii) s: the 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 the MCUBA is $O(\Delta s)$, that of the DS and DSTA is $O(\Delta)$. From our simulations, we found the average link quality of the DSTA as to be W, DS to be $2.3W$, MCUBA to be $7W$ as shown in Section 7.2. Let α, β and γ be the number of retransmissions for a successful packet transmission needed by the DSTA, DS and MCUBA. From Lal's study [15], we may state that $\alpha < \beta < \gamma$ with regard 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 the MCUBA (M), DS (D) and DSTA (T) are given by

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

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

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

It is obvious from Equations (2) and (3) that D is smaller than T. We now compare M and D. From Equations (2) and (1), $D - M$ is given by

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

From Equation 4, as the number of events increases, the MCUBA consumes less energy than the DS. For $\beta > s\alpha$, the MCUBA consumes less energy than the 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.

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

The MCUBA is the enhanced version of 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 the 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 and 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 the 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 MASC. The energy consumption of MASC can be reduced by the overhearing technique as applied in the 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. Hence rotation of the cluster heads may be provided by the clustering approaches. The 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 MASC and the 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 the MCUBA with minor modifications.

The periodic running of the clustering algorithms is not the only way of supporting fault tolerance. One other method is the selection of backup cluster heads [2]. The 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 the *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 the *MEMBER* or *MEMBER_END* state receives a *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 the *WT_LEADER* state. When the cluster head in the *LEADER_END* state receives the *POLL_LEADER* message, it replies with a *HEARTBEAT* message. If the cluster head does not respond, the backup cluster head tries to send a *POLL_LEADER* message for *poll_number* times defined by the user. When the backup realizes that the cluster head malfunctions, a *LEADER_DEAD* event occurs in the 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 the 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 heads or cluster members 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

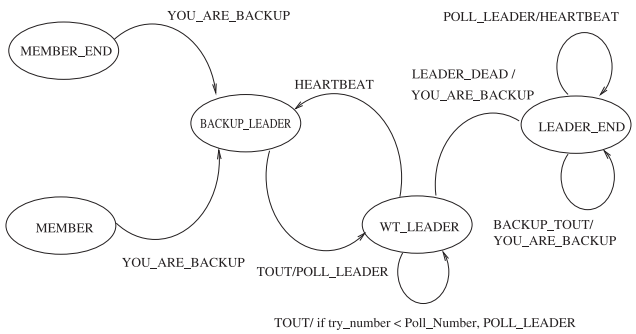


FIGURE 29. FSM for providing fault tolerance.

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 the 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 networks in terms of the 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 the MCUBA, by the 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 a 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-theoretic approaches. The energy consumed in MASC is reduced in the MCUBA by applying the 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. Wirel. Commun.*, **1**, 660–670.

[2] Younis, O. and Fahmy, S. (2004) Distributed Clustering in Ad-hoc Sensor Networks: A Hybrid, Energy-Efficient Approach. *Proc. INFOCOM'04*, Hong Kong, March 7–11, 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. *Proc. MASS'05*, Washington, November 7–10, 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. *Comput. Netw.*, **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. *Proc. ICCS'08*, Krakow, June 23–25, pp. 519–528. Lecture Notes in Computer Science 5101. Springer, Berlin.

[7] Chen, Y.P. and Liestman, A.L. (2002) Approximating Minimum Size Weakly-Connected Dominating Sets for Clustering Mobile Ad Hoc Networks. *Proc. MOBICOM'02*, Atlanta, September 23–28, 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. Parallel Distrib. Syst.*, **13**, 14–25.

[9] Wu, J., and Li, H. (1999) A dominating-set-based routing scheme in ad hoc wireless networks. *Telecommun. Syst.*, **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. Parallel Distrib. Syst.*, 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, 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. *Proc. ICCSA'06*, Glasgow, May 8–11, pp. 571–578. Lecture Notes in Computer Science 3991. Springer, 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. *Proc. GCC'03*, Shanghai, December 7–10, pp. 719–722. Lecture Notes Computer Science 3032. Springer, Berlin.

[14] Banerjee, S. and Khuller, S. (2001) A Clustering Scheme for Hierarchical Routing in Wireless Networks. Technical Report CS-TR-4103, University of Maryland, Annapolis, MD, 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. *Proc. GLOBECOM'03*, San Francisco, December 1–5, Vol. 1, pp. 446–452. IEEE, Washington.

[16] West, D. (2001) *Introduction to Graph Theory*, (2nd edn). Prentice-Hall, NJ.

[17] Bunke, H. (2000) Graph Matching: Theoretical Foundations, Algorithms, and Applications. *Proc. VI'03*, Montreal, June 11–13, 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. *Proc. DIALM-POMC'04*, Philadelphia, October 1, pp. 17–23, ACM, New York.

[21] Clark , B.N., Colbourn, C.J. and Johnson, D.S. (1990) Unit disk graphs. *Discrete Math.*, 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. Sensor. 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 Trans.*, 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. *Proc. STFSSD'09*, Tokyo, March 17, pp. 210–214. IEEE, Washington.
[25] Polastre, J., Hill, J. and Culler, D. (2004) Versatile Low Power Media Access for Wireless Sensor Networks. *Proc. SENSYS'04*, Baltimore, November 3–5, 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. *Proc. SENSYS'03*, Los Angeles, November 5–7, 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. *J. Discrete Algorithms*, 4(4), 588–607.
[29] cs.DC/0410047. (2004) Simple distributed weighted matchings. Nijmeng Institute for Computing and Information Sciences, Radbound University, Nijmegen, The Netherlands.
[30] Chattopadhyay, S., Higham, L. and Seyffarth, K. (2002) Dynamic and Self-Stabilizing Distributed Matching. *Proc. PODC'02*, Montemery, July 21–24, pp. 290–297. ACM, New York.
[31] Karaata, M. and Saleh, K. (2000) A distributed self-stabilizing algorithm for finding maximal matching. *Comput. Syst. Sci. Eng.*, 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 . *Proc. ICDCSW'04*, Tokyo, March 23–24, Vol. 7, pp. 505–510. IEEE, Washington.
[33] Gabow, H. (1990) Data Structures for Weighted Matching and Nearest Common Ancestors with Linking. *Proc. SODA'90*, Vancouver, January 23–25, pp. 434–443. ACM, New York.
[34] Avis, D. (1983) A survey of heuristics for the weighted matching problem. *Networks*, **3**, 475–493.
[35] Preis, R. (1999) Linear time 1/2-approximation algorithm for maximum weighted matching in general graphs. *ACM Trans. Algorithms*, **1**, 107–122.
[36] Drake, D. and Hougardy, S. (2003) Improved Linear Time Approximation Algorithms for Weighted Matchings. *Proc. APPROX'03*, Princeton, August 24–26, pp. 14–23, Lecture Notes Computer Science 2764. Springer, Berlin.
[37] Pettie S. and Sanders, P. (2004) A simple linear time 2/3-ϵ approximation for maximum weight matching. *Inform. Process. Lett.*, 91(6), 271–276.
[38] Uehara, R. and Chen, Z. (2000) Parallel approximation algorithms for maximum weighted matching in general graphs. *Inform. Process. Lett.*, 76, 13–17.
[39] Wattenhofer, M. and Wattenhofer, R. (2004) Distributed Weighted Matching. *Proc. DISC'04*, Amsterdam, October 4–7, pp. 335–348, Lecture Notes Computer Science 3274. Springer, Berlin.
[40] Lotker, Z., Patt-Shamir, B. and Pettie, S. (2008) Improved Distributed Approximate Matching. *Proc. SPAA'08*, Munich, June 14–16, pp. 129–136. ACM, New York.
[41] Nieberg, T. (2008) A Local Approximation Algorithm for Maximum Weight Matching. *Proc. CTW'08*, St. Croix, May 11–14, pp. 44–47. IEEE, Washington.
[42] Ganeriwal S., Kumar, R. and Srivastava, M. B. (2003) Timing-Sync Protocol for Sensor Networks. *Proc. SENSYS'2003*, Los Angeles, November 5–7, 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*. *Proc. SENSYS'03*, Los Angeles, November 5–7, pp. 14–27. ACM, New York.
[46] Dagdeviren, O. and Erciyes, K. (2006) A Distributed Backbone Formation Algorithm for Mobile Ad Hoc Networks. *ISPA'06*, Sorrento, December 4–6, pp. 219–230, Lecture Notes Computer Science 4330. Springer, Berlin.

APPENDIX

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

Algorithm 1. MASC for node$_j$ receiving message from node$_i$.

initially cur_state_j: the current state of node$_j$
    $cur\_state \leftarrow$ LEADER
    $round \leftarrow 0$
    msg: the received message, includes: C_i, RSS_i, $leader_i$
    C_i: the set of nodes in the same cluster with node$_i$
    RSS_i: the second largest RSS value of a cluster head node$_i$, used to determine the leadership
    Γ_j: the set of neighbors of node$_j$ which do not belong to the cluster of node$_j$, given as the parameter
    $M_w(A)$: the operator which returns the id of the node in set A which has the largest weight of the edge connecting to node$_j$
    $candidate_j \leftarrow M_w(\Gamma_j)$
    m_rounds: number of maximum rounds, given as the parameter
    R: the set of collected requests
    Legend : □ State ∧ $input\_message \longrightarrow$ actions
procedure MAIN(Γ_j, m_rounds)
  loop
    □ LEADER ∧ $Request \longrightarrow$
      call RQST(msg, m_rounds,Γ_j)
    □ LEADER ∧ $New\_Round\_Interrupt \longrightarrow$
      $R \leftarrow \emptyset$
      $candidate_j \leftarrow M_w(\Gamma_j)$
      send REQUEST to $candidate_j$
      $cur\_state_j \leftarrow CANDIDATE\_WT\_ACK$
    □ LEADER ∧ $Drop \longrightarrow$

```
            call DRP (msg,m_rounds,Γ_j)
        □ MATCHED_LEADER ∧ Request ⟶
            send DROP to node_i
        □ MATCHED_LEADER ∧ End_Of_Round ⟶
            cur_state_j ← LEADER
        □ CANDIDATE_WT_ACK ∧ Request ⟶
            call RQST(msg,m_rounds,Γ_j)
        □ CANDIDATE_WT_ACK ∧ Request_Ack ⟶
            call CLSTR (msg,m_rounds,Γ_j)
        □ CANDIDATE_WT_ACK ∧ Drop ⟶
            call DRP (msg,m_rounds,Γ_j)
        □ MEMBER ∧ Request ⟶
            if id_i ∈ C_j then
                send Request_Ack
            else forward Request to leader_j
            end if
    end loop
end procedure
```

Algorithm 2. MASC for node$_j$ receiving message from node$_i$ (Part 2).

```
procedure RQST(msg,m_rounds,Γ_j)
    if leader_i = id_j then
        send Request_Ack
    else if id_i=candidate_j then
        call CLSTR(msg, m_rounds,Γ_j)
    else
        R ← R ∪ id_i
    end if
end procedure
procedure CLSTR(msg, m_rounds, Γ_j)
    send Request_Ack
    if (RSS_i>RSS_j)or((RSS_i=RSS_j) and id_i>id_j) then
        cur_state_j ← MEMBER
        leader_j ← id_i
    else
        if round < m_rounds then
            round ← round + 1
            set Round_Timer
        end if
        cur_state_j ← MATCHED_LEADER
        Γ_j ← Γ_j ∪ Γ_i
    end if
end procedure
procedure DRP(msg,m_rounds, Γ_j)
    Γ_j ← Γ_j / id_i
    candidate_j ← M_w(Γ_j)
    if candidate_j = ∅ then
        if round < m_rounds then
            round ← round + 1
            set Round_Timer
        end if
        cur_state_j ← MATCHED_LEADER
    else
        if candidate_j ∈ R then
            call CLSTR(msg_candidate_j,m_rounds,Γ_j)
        end if
    end if
end procedure
```

Algorithm 3. MCUBA for node$_j$ receiving message from node$_i$.

```
initially cur_state_j: the current state of node_j
    cur_state ← LEADER
c_level: the cluster level
    c_level ← 0
uc_level: the upper cluster level
b_candidate: the backbone candidate
    msg: is the received message, includes: C_i, RSS_i, leader_i
    C_i: the set of nodes in the same cluster with node_i
    RSS_i: the second largest RSS value of a cluster head node_i,
used to determine the leadership.
    Γ_j: the set of neighbors of node_j which do not belong to the
cluster of node_j
    Γ_b: the set of neighbors of sink node which do not belong to
the backbone
    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 node_j.
```

Algorithm 4. MCUBA for node$_j$ receiving message from node$_i$ (Part 2).

```
    R: the set of collected Request msgs.
    BR: the received Backbone_Request msg.
    D: the set of collected Drop msgs.
    Legend : □ State ∧ input_msg ⟶ actions
procedure MAIN(Γ_j, uc_level)
    loop
        □ LEADER ∧ Request ⟶
            call RQST(msg,uc_level,Γ_j)
        □ LEADER ∧ Ldr_TOUT ⟶
            call LDR(msg,uc_level,Γ_j)
        □ LEADER ∧ Drop ⟶
            call DRP(msg,uc_level,Γ_j)
        □ LEADER ∧ Backbone_Request ⟶
            BR ← msg
        □ LEADER_END ∧ Request ⟶
            call RQST_END(msg)
        □ LEADER_END ∧ Backbone_Request ⟶
            send Backbone_Request_Ack to node_i
        □ CANDIDATE_WT_ACK ∧ Request ⟶
            call RQST(msg,uc_level,Γ_j)
        □ CANDIDATE_WT_ACK ∧ Drop ⟶
            call DRP(msg,uc_level,uc_level)
        □ CANDIDATE_WT_ACK ∧ Request_Ack ⟶
            call CLSTR(msg,uc_level,uc_level)
        □ CANDIDATE_WT_ACK ∧ Backbone_Request
            ⟶ BR ← msg
```

```
39:          □ MEMBER ∧ Request ⟶
40:            if id_i ∈ C_j then
41:              send Request_Ack to node_i
42:            else
43:              forward Request to leader_j
44:            end if
45:          □ MEMBER ∧ Backbone_Request ⟶
46:            forward Backbone_Request to node_j
47:          □ MEMBER_END ∧ Request ⟶
48:            call RQST_END(msg)
49:          □ MEMBER_END ∧ Backbone_Request ⟶
50:            forward Backbone_Request to leader_j
51:          □ SINK_WT_FOR_BACKBONE ∧ Request ⟶
52:            send Drop to node_i
53:          □ SINK_WT_FOR_BACKBONE ∧
54:            Backbone_Request_Ack ⟶
55:            call BCKBN(msg,Γ_j)
56:      end loop
57:  end procedure
58:  procedure RQST(msg, uc_level,Γ_j)
59:      if leader_i = id_j then
60:          send Request_Ack
61:      else if id_i=candidate_j then
62:          call CLSTR(msg, uc_level,Γ_j)
63:      else
64:          R ← R ∪ 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,Γ_j)
68:      send Request_Ack
69:      leader_j ← id_i
70:      cluster_level_j ← 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 >= uc_level then
75:              cur_state_j ← MEMBER_END
76:          else
77:              cur_state_j ← MEMBER
78:          end if
79:      else
80:          Γ_j ← Γ_j ∪ Γ_i
81:          candidate_j ← M_w(Γ_j)
82:          if cluster_level_j >= uc_level and candidate_j ≠ ∅ then
83:              cur_state_j ← CANDIDATE_WT_ACK
84:          else
85:              send Drop to all node ∈ R
86:              R ← ∅
87:              if id_j = sink_j then
88:                  Γ_b ← Γ_j
89:                  b_candidate_j ← M_w(Γ_b)
90:                  cur_state_j ←
91:  SINK_WT_FOR_BACKBONE
92:                  if b_candidate_j ≠ ∅ then
93:                      send Backbone_Request to
94:  b_candidate
95:                  else
96:                      cur_state_j ← SINK_END
97:                  end if
98:              else
99:                  cur_state_j ← LEADER_END
100:                 if BR ≠ ∅ 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 ∈ C_j then
109:         send Request_Ack to node_i
110:     else
111:         send Drop to node_i
112:     end if
113: end procedure
114: procedure BCKBN(msg,Γ_j)
115:     Γ_b ← Γ_b ∪ Γ_i
116:     b_candidate_j ← M_w(Γ_b)
117:     if b_candidate_j ≠ ∅ then
118:         send Backbone_Request to b_candidate_j
119:     else
120:         b_candidate_j ← 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,Γ_j)
124:     Γ_j ← Γ_j / id_i
125:     candidate_j ← M_w(Γ_j)
126:     if candidate_j = ∅ then
127:         cur_state ← MATCHED_LEADER
128:     else
129:         if candidate_j ∈ R then
130:             call CLSTR(msg_candidate_j, uc_level,Γ_j)
131:         end if
132:     end if
133: end procedure
134: procedure LDR(msg,uc_level,Γ_j)
135:     if cluster_level < uc_level then
136:         Request to candidate_j
137:         cur_state ← CANDIDATE_WT_ACK
138:     else
139:         send Drop to all node ∈ R
```

```
140:        if id_j ≠ id_sink then
141:            cur_state_j ← LEADER_END
142:            if BR ≠ ∅ then
143:                send Backbone_Request_Ack to BR_i
144:            end if
145:        else
146:            Γ_b ← Γ_j
147:            b_candidate_j ← M_w(Γ_b)
148:            cur_state_j ←
149: SINK_WT_FOR_BACKBONE
150:            if b_candidate_j ≠ ∅ then
151:                send BACKBONE_REQUEST to b_candidate
152:            else
153:                cur_state_j ← SINK_END
154:            end if
155:        end if
156:    end if
157: end procedure
```