Novel Broadcast/Multicast Protocols for Dynamic Sensor Networks*

Wei Chen[1], A.K.M.Muzahidul Islam[2], Mohan Malkani[1], Amir Shirkhodaie[1], Koichi Wada[2], Mohamed Zein-Sabatto[1]

1. Tennessee State University, USA (wchen, mmalkani, ashirkhodaie, mzein@tnstate.edu)

2. Nagoya Institute of Technology, Japan (wada@ nitech.ac.jp, islam@phaser.elcom.nitech.ac.jp)

Abstract: In this paper, we have proposed a time efficient, energy saving and robust broadcast/multicast protocol for reconfigurable cluster-based sensor network. In our broadcast protocol, a broadcast can be executed in $O(hd^2 + D^2)$ rounds and each node needs to be awake in $O(D^2)$ rounds, where D and d are the degrees of *G* and the sub-network induced by the network backbone, respectively, and *h* is the height of the backbone. When *k* channels are available, the broadcast can be executed in $O((hd^2 + D^2)/k)$ rounds and each. We show that our broadcast protocol can be readily modified to the one for multicast. The cluster-based architecture used in this paper for a sensor network is an improved version in [19]. The proposed network architecture is self-constructible and self-reconfigurable by using two topological management operations: *node-move-in* and *node-move-out*. Details of the protocol along with experimental results are discussed. Simulation results show that the protocol performance is much better than that in the theoretical analysis.

1. Introduction

Wireless sensor networks (WSNs) are clear application specific and have specialized communication patterns in which broadcast, multicast and data gathering are more important than traditional point-to-point communication in computer networks. The geographical topology of a WSN changes when network connectivity changes. For example, a power-trained sensor node withdraws its connection from its network when its battery voltage is low and comes back to the network when it is recharged. When the topology of a WSN is changed as such, the routing protocol and network architecture are necessary to be updated.

Broadcast protocols have been well studied in WSNs. Given a flat WSN, say *G* (an unstructured WSN formed naturally after sensor nodes are deployed) with *n* sensor nodes, assuming that the nodes know only their IDs and use a single radio channel without collision detection capability, the lower bound of a broadcast is $\Omega(n)$ rounds [1]. An $O(n)$ broadcast is achieved in [9]. If the nodes have topology awareness, the lower bound of a broadcast can be reduced to $\Omega(\log^2 n)$ rounds[1]; and in this case an $O(L\log^2 n)$ [8] and an $O(L + \log^5 n)$ [12] broadcast are achieved, where *L* is the diameter of *G*. Though *L* is much smaller than *n,* it is expensive to maintain an entire network knowledge in each node for a dynamic WSN. In addition to the above deterministic broadcast some randomized broadcast protocols have been also proposed [3, 6, 10, 16, 21]. The typical approach in these protocols is to combine a flooding approach with some heuristic and genetic techniques to avoid broadcast storm problems caused by flooding.

A hierarchically organized sensor network usually offers much better networking performance. Clustering has been used to induce a hierarchical structure over a flat WSN which minimizes communication overhead, facilitates energy efficient sensing and networking operation, and facilitates network self-reconfiguration. The basic idea is that of breaking the network into clusters which are smaller in scale and usually simpler and more efficient to be managed by the node called as *cluster head*. By using clustering induced hierarchy, a subsequent *backbone* is formed consisting of cluster heads and gateway nodes which serve as communication relays between the adjacent clusters. For minimizing the overhead of clustering and the structure maintenance, in many proposed approaches cluster head nodes are selected through finding a small *dominating set* (*DS*) or a large *independent set* (*IS*) of *G* [4, 5, 7, 11, 13, 20, 22]. Finding a minimum *DS* or an optimal *IS* of *G* is an *NP*-complete problem. In a complete hierarchical cluster-based structure, a cluster is formed by the cluster head connected with its members and the backbone communication route is formed by joining cluster heads through gateway nodes. A cluster-based structure can be constructed from $O(n)$ to $O(n^2)$ rounds depending on how the structures of cluster and backbone are specified. An asymptotically optimal maximum *IS* can be found in polylogarithmic rounds by using randomized algorithms [13, 14]. After a cluster-based structure is built for a WSN, sustaining its network topology is crucial.

Recently, several reconfigurable cluster-based architectures have been proposed in the literature. In [17], adaptive and reconfigurable overlays for multi-scale communication in WSNs are proposed, where the scale of a cluster called as cells is adaptable. However, the broadcast and other network functions are discussed only inside cells, and the approaches depend on conflict-free MAC protocols

* This work is supported in part by the grant from the ARO W911NF-04-2-0049, U.S.A., the Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research(C) 17500036, and Scientific Research on Priority Areas New Horizons in Computing (C08).

[2, 18]. In addition, in the broadcast inside a cell, each node needs to be awake until all neighbors received the broadcast message. In [19], another reconfigurable cluster-based structure is proposed. It consists of at most p clusters with a backbone tree of at most $2p-1$ nodes, where the head form a dominating set of G, and p is not larger than the smallest number of complete sub-graphs in G. A broadcast in G can be executed in at most $4p-2$ rounds. In the broadcasting mode, each node needs to be awake until all its neighbors received the source data.

In this paper, we present time efficient, power saving and robust broadcast/multicast protocols for an improved cluster-based network structure as in [19]. Our broadcast can be executed in $O(hd^2 + D^2)$ rounds, where D is the maximum degree of the nodes in G, d is the maximum degree of the nodes in the graph induced by the backbone, and h is the height of the backbone. The broadcasting protocol in [19] is based on depth-first-order on the backbone. Namely, only one node is allowed to relay the source message at each round. Therefore, the broadcast will be unsuccessful if a node/link failure happens. Our broadcasting protocol is based on a *collision-free-flooding* approach on the backbone which is independent to MAC protocol. In the new broadcasting scheme, more than one node can relay the source message at each round if the nodes do not cause collisions. This approach is more robust. For example, even some nodes fail to relay the broadcast message, which may cause a partial part of the network fails to receive the message, other nodes can still relay the message to the remaining part of the network. We will show that the new broadcast is also more energy efficient that is each node needs to be awake only in $O(D^2)$ rounds. A backbone usually is much smaller than G, therefore, d and h are small which means that our broadcast is fast. Our approach can be readily modified to one for k radio channels in which a broadcast can be executed in $O((hd^2 + D^2)/k)$ rounds and each node in the broadcast needs to be awake only in $O(D^2/k)$ rounds. The proposed collision-free-flooding approach can be also used for the multicasting purposes. In our cluster-based WSN, each node needs to know a bit more information about network topology than that in [19]; however self-construction and self-reconfiguration can be still achieved efficiently. Simulation results show that not only in the worst case but also in the average case the performance of our protocols are much more time and energy efficient than in [19].

2. Cluster-Based Structure

Let a WSN be represented by an undirected graph $G = (V, E)$, where V is the set of nodes and E is the set of edges (Fig. 1 (a)). In G, nodes u and v have an edge between them *iff* they are in the transmission range with each other. In this section, we define a cluster-based structure of G with some nice properties. Self-initialization and self-reconfiguration of the structure will be discussed in Section 5.

In our clustering, the nodes of G are partitioned into node-disjoint clusters. There is one head node in each cluster which connects to all other member nodes. In any two neighbored clusters, there is a gateway node which is the member of one cluster but connects to the head nodes of both clusters (Fig.1 (b)). The following definition gives a precise construction of the clustering.

Definition 1 Given a graph $G = (V, E)$ with a specified node r, a cluster-based structure of G, called as *cluster-net* of G and denoted as *CNet*(G), is a spanning tree of G with root r. In *CNet*(G), each node knows its status either as *cluster-head,* or as *gateway*, or as *pure-member*. The structure of *CNet*(G) is defined recursively as follows:

(1) If G consists of only one node r, then r is the root of *CNet*(G) and r is a cluster-head.

(2) Let $G_{old} = (V, E)$ be a graph with n ($n \geq 1$) nodes, and its cluster-net be $CNet(G_{old}) = (V, E_{CNet})$ .Assume that $G = (V \cup \{new\}, E \cup E')$ is a graph obtained by adding a node *new* to G_{old}, and $E' = \{(new, u) \mid u \in V$, and *new* and u are in the transmission range with each other$\}$. The cluster-net of G is defined as *CNet*(G) = $(V \cup \{new\}, E_{CNet} \cup \{(new, w)\})$, where w is the parent of *new* in *CNet*(G). Let U be the set of the nodes in V connected to *new*. Node w and the status of the nodes in *CNet*(G) are decided by the following rules:

(i) The nodes of G other than *new* and w have the same status as they have in $CNet(G_{old})$,

(ii) Node w and the status of *new* and w are decided as follows: If there exist cluster-heads in U, select one of them to be w (based on the criteria an application needs, such as on energy level). In this case, w remains as a cluster-head and *new* is set to be a pure-member of w (Fig.2 (a)). Else if there exist gateways in U, select one of them to be w. In this case, w remains as a gateway and *new* is set to be a cluster-head of a new cluster (Fig.2(b)). Else, U contains only pure-members. In this case, select one of them to be w; then set w to be a gateway and *new* to be a cluster-head (of a new cluster) (Fig.2(c)).

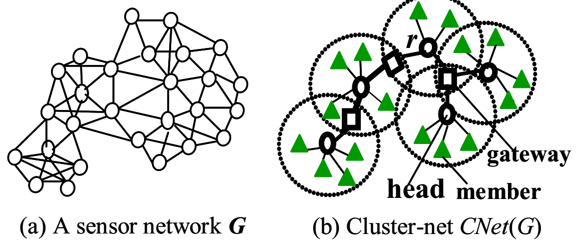

Fig. 1 A sensor network G, and its cluster-net *CNet*(G) and backbone tree (formed by cluster-heads and gateways)

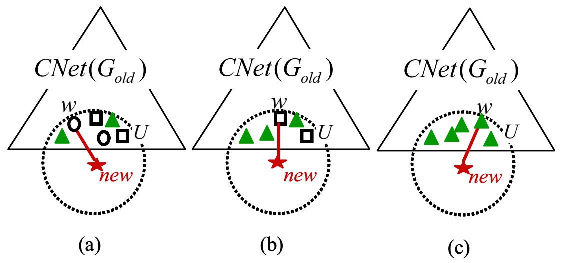

Fig.2 (a) *new* is connected to a cluster-head w and *new* will be a *pure-member* of w; **(b)** *new* is connected to a gateway w and *new* will be a *cluster-head* of a new cluster; **(c)** *new* is connected to a pure-member w; w will be changed to a *gateway* and *new* it will be a *cluster-head* of a new cluster.

Definition 2 Given a graph G and its cluster-net *CNet*(G), a backbone of *CNet*(G), denoted as *BT*(G), is a sub-tree of *CNet*(G) formed by cluster-heads and gateways and it has the same root as *CNet*(G) (Fig. 1(b))

The root of *CNet*(G) can be considered as a sink in a WSN. It can transreceive the information to and from a base stations. In order to boost the robustness of the proposed structure strongly robust, more than one cluster-net may be selected in the same way from different roots (sinks) so that if one cluster-net fails others can still be used.

Property 1 Assuming that G has n nodes and p is the smallest number of the complete sub-graphs in G, $CNet(G)$ and $BT(G)$ have the following properties [19]: (1) $CNet(G)$ has at most p clusters and $BT(G)$ has at most $2p$-1 nodes, (2) there is no edge between cluster heads in G, and (3) when G is a unit disk graph (G is a unit disk graph *iff* any two nodes in G can transmit with each other when their distance is not larger than one unit), the number of the clusters in $CNet(G)$ is not larger than $5 \times |\mathrm{MDS}|$, where *MDS* is the minimum *DS* of G.

Let the depth of the root to be null. According to Property 1(2), the nodes of $BT(G)$ in depth i are cluster-heads if i is even and they are gateways if i is odd (Fig.1 (b)).

3. Broadcast and Multicast

In this section, we primarily presented a detail of the sensor network model and our broadcasting and multicasting protocols by Collsion-Free Flooding. We compared our protocols with the protocol by Depth-First-Order in [19].

3.1 Model of Sensor Networks

In this paper, the model of a flat (unstructured) sensor network G is as follows: (1) all nodes use a single radio channel; (2) each node has a distinct ID number and it has no other network knowledge (e.g., neighbors' IDs, diameter of the network, number of nodes in the network, etc.); (3) each node repeats transmission or reception and performs its local computation in a fixed interval, called *round.* In each round, a node acts as either a transmitter or a receiver; and (4) nodes have no collision detection, i.e., a node that acts as a receiver will get a message in a given round *iff* there is exactly one of its neighbors that transmits in this round.

3.2 Broadcast by Depth-First-Order

In [19], the nodes of cluster-net $CNet(G)$ were built from a flat network G under assumption that each node knows its neighbors' IDs. In order to transmit a broadcast message to all nodes of G from the source node, the broadcast message is relayed on backbone tree $BT(G)$ in a depth-first order. In other words, the message travels an *Eulerian* tour in $BT(G)$ by replacing every undirected edge in $BT(G)$ with two edges in opposite directions. In the tour, each node of $BT(G)$ transmits the message at least once, and at each round only one node transmits the message. Therefore, when the tour finishes, all nodes of G have received the message without collision. In general, the transmission tour of a message m from a source node s to all other nodes in a tree T can be described as a procedure *Eulerian*(s, T, m) as follows.

Let v (at the beginning s) be the node with a token for relaying m. First, v selects a node u from v's neighbors to whom v has not send m yet, and then transmits m with u's ID. When u received m with u's ID, it got the token and it will relay m at next round. Other neighbors of v will discard the message when they received it. If v has already transmitted m to all its neighbors, v will pass the token to its parent (i.e., it transmits m with its parent's ID) which is the node v received m first from. This procedure will repeat until the token turns back to source node s. In the tour, each node transmits m exactly the times of its degree in T. In other words, m is relayed on each edge of T exactly twice.

According to Property 1(1), $BT(G)$ has at most $2p-1$ nodes, therefore, the broadcast *Eulerian*(s, $BT(G)$),m) can be completed in at most $4p-2$ rounds.

3.3 Broadcast by Collision-Free Flooding

Our broadcast in G is executed by flooding the broadcast message on $CNet(G)$ from one depth to next depth starting at the root. If the source node is not the root, it will transmit the source message along the path back to the root using at most h rounds, where h is the height of $CNet(G)$. To avoid collision in the flooding, time division mechanism (TDM) is used: each internal node in $CNet(G)$ is assigned with a time-slot numbered from 1 to Δ such that if nodes at depth i transmit a message at the assigned time-slots then nodes at depth i+1 will receive the message without collision. The flooding will stop at the leaves of $CNet(G)$.

We assume that $CNet(G)$ is constructed (in Section 4 and Section 5) with following knowledge: each internal node v in $CNet(G)$ knows its depth and transmission time-slot *v.time-slot* numbered from 1 to Δ (Δ will be determined later), and r knows it is the root and knows Δ. The transmission time-slots assigned to the nodes have to meet the following condition:

***Time-Slot Condition 1*:** (1) For each node v of $CNet(G)$ at depth i+1, assuming that P_v is the set of the nodes at depth i who are connected with v by the edges of G, there is at least one node in P_v whose transmission time-slot is different from those of the other nodes in P_v.

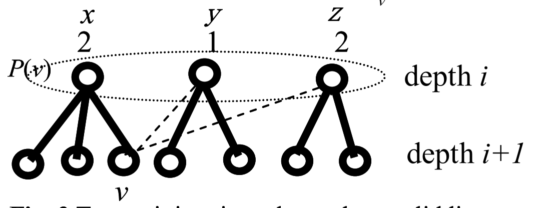

Fig. 3 Transmitting time-slots, where solid lines are the edges of $CNet(G)$, and dot lines are edges of G which are not in $CNet(G)$

In Fig. 3, P_v={x, y, z}. The transmission time-slot of y is 1 and it is different from the time-slots of x and z.

```
Algorithm1 CollisionFreeFlooding(CNet(G), $\Delta$,m)
Root r : r transmits package (m, r.time-slot, $\Delta$,0) at round r.time-slot, where m is the broadcast message.
Other node v: If v is an internal node at depth i+1 $(0 \le i \le h-2)$ and it received package (m, t, $\Delta$, i), v wait $\Delta - t$ rounds, and then transmits (m, v.time-slot, $\Delta$,i+1) at round v.time-slot.
```

Lemma 1 By using $CNet(G)$, a broadcast can be completed in G in $\Delta \cdot (h-1)$ rounds, where Δ is the largest transmission time-slot in $CNet(G)$, h is the height of $CNet(G)$. In the broadcast, each node needs to be awake in 2Δ rounds.

Proof: In Algorithm 1, node v at depth i+1 needs to wait $\Delta - t$ rounds after it received the message (m, t, Δ, i) in order to let all nodes at depth i finish transmission. Then, v transmits (m, *v.time-slot*, Δ, i+1) at time-slot *v.time-slot*. Therefore, each node needs at the most 2Δ rounds in woken up for receiving and transmits the message. According to *Time-Slot Condition* 1, each node in $CNet(G)$ can receive the broadcast message at least at one time-slot without collision. The transmission from one depth to the next depth can be finished in Δ rounds. □

Algorithm 1 has some other nice properties.

Relaxation of Synchronization: To avoid collision, time-slots (or rounds) are synchronized in Algorithm1. However, the synchronization is not needed among all nodes. It is clear to see that only the nodes at the same depth need to be tightly synchronized.

Robustness: In the broadcast by depth-first-order [19] in Section 3.2, at each round only one node relays the broadcast message. If one node or link fails, the whole broadcast in Eullerian tour will stop. However, in the broadcast by collision-free-flooding, more than one node at the same depth are allowed to relay the message at the same round. Even some nodes or links fail, the others who got the message will continue to relay it in the network.

Multi-Channels: In Algorithm 1, we use a single radio channel. If k channels are available, then the nodes at the same depth which are assigned by transmission time-slots i+1, i+2, …, i+k, where $0 \le i \le \lfloor \Delta / k \rfloor$, can transmit the message at same time-slot using k different channels, Therefore, the broadcast can be completed in $\Delta h / k$ rounds and each node needs to be awake in $2\Delta / k$ rounds.

In the remaining of the section, we focus on improving Algorithm 1. Let $G(V_{BT})$ = (V_{BT}, F) be the subgraph of G induced by V_{BT}, where V_{BT} is the set of nodes in BT(G) and F is the set of all edges in G with both ends in V_{BT}. According to Lemma 1, by using BT(G) in Algorithm 1, a broadcast in $G(V_{BT})$ can be completed in $\delta(h-1)$ rounds, where δ is the largest transmission time-slot assigned to the internal nodes of BT(G). Since BT(G) contains only cluster heads and gateways, it is much smaller than $CNet$(G). Therefore, δ is much smaller than Δ. In the following improved algorithm, the broadcast message m is first flooded into the nodes of BT(G) in $G(V_{BT})$ by executing *CollisionFreeFlooding*(BT(G), δ ,m), and then into the leaves of $CNet$(G). Two types of transmission time-slots are assigned to the internal nodes of $CNet$(G): *b-time-slot* is used for flooding the message to the nodes of BT(G), and *l-time-slot* is used for transmitting the message to the leaves of $CNet$(G). They need to satisfy the following condition:

Time-Slot Condition 2: For each internal node (leaf, resp.) v of $CNet$(G) at depth i+1, assuming that P_v is the set of the nodes at the depth i who are connected with v by the edges of G, there is at least one node in P_v whose b-time-slot (l-time-slot, resp.) is different from those of the other nodes in P_v.

We assume that the nodes of BT(G) know their depth and their *b-time-slots* and *l-time-slots*, and the root know the height of $CNet$(G) and δ.

Algorithm2
ImprovedCollisionFreeFlooding ($CNet(G)$, δ ,m)
(Step 1) The root calls *CollisionFreeFlooding* ($BT(G)$, δ , (m,h)) to broadcast in $G(V_{BT})$ the message m with and the height h of $CNet$(G). In the algorithm, each node uses its *b-time-slot* as the transmission time-slot.
(Step 2) For any node w of BT(G) at depth i ($1 \le i \le h-1$), if w received the message sent out by a node u at u's *b-time-slot* t, w waits $\delta(h-i-1)-t$ time-slots until Step 1 finishes, and then transmits m at its *l-time-slot.*

Theorem 1 (1) A broadcast in G can be completed in $\delta h + \Delta$ rounds. (2) Each node needs to be awake in $2\delta + \Delta$ rounds. (3) If k channels are available, a broadcast can be completed in $(\delta h + \Delta)/k$ rounds and each node needs to be awake in $(2\delta + \Delta)/k$ rounds.

Proof: We first prove conclusion (1). In the first step of Algorithm 2, the broadcast in $G(V_{BT})$ is executed on BT(G). According to Lemma 1, the broadcast can be finished in $\delta \cdot h$ rounds. When a node received the broadcast message in Step 1, it waits until Step 1 finishes; then it transmits the message at its *l-time-slot* which is not larger than Δ . Therefore, the transmission in Step 2 can be completed in Δ rounds.

Now we prove conclusion (2). According to Lemma 1, in Step 1 the nodes in BT(G) need to be awake in 2δ rounds. In Step 2, node w of BT(G) needs to wait $\delta(h-i-1)-t$ rounds until Step 1 finishes; then it transmits the broadcast message to the leaves of $CNet$(G) at its *l-time-slot.* In order to save the energy, w goes to sleep-mode after it relayed the message in BT(G) in Step 1; then it wakes up after $\delta(h-i-1)-t$ rounds. It needs to wake up when Step 1 finished and then transmits the message at its *l-time-slot*. Therefore, w needs to be awake at most Δ rounds in Step 2. The proof of conclusion (3) for k channels is similar to that for Algorithm 1. □

In Section 4, we will prove that *b-time-slots* and *l-time-slots* are not larger than D(D+1)/2+1 and d(d+1)/2+1, respectively, where D and d are the degrees of the sensor network G and $G(V_{BT})$, respectively. Relaxation of synchronization and robustness for Algorithm 2 can be discussed in the same way as that we have done for Algorithm 1. In Algorithm 2, before starting Step 2 the internal nodes of $CNet$(G) needs to wait certain rounds until Step 1 finishes. To relax the synchronization among the waiting rounds, the nodes can take a synchronization before they transmit the message to the leaves of $CNet$(G).

3.4 Multicast

A multicast is the broadcast to a group of specified nodes. In a multicast, in addition to the group nodes, some other nodes are needed to relay the broadcast message. Let G have k groups and each node has a group list indicating which groups it belongs to. Our broadcast protocols can be readily modified to the one for multicast using relaying nodes as least as possible.

Let us consider a cluster-based structure for multicast, denoted as $MCNet$(G). In $MCNet$(G), in addition to all the properties that $CNet$(G) has, each node maintains a *group-list*, and the internal nodes of $MCNet$(G) keep one more list called *relay-list*, where a node v has f in its group-list if v belongs to group f, and it has g in its relay-list if v has a descendant in $MCNet$(G) belonging to group g. Fig.4 shows an example with two groups. In the figure, there are two lists at each node, the upper one is the *group-list* and the lower one is the *relay-list*. The construction and reconfiguration of $MCNet$(G) will be

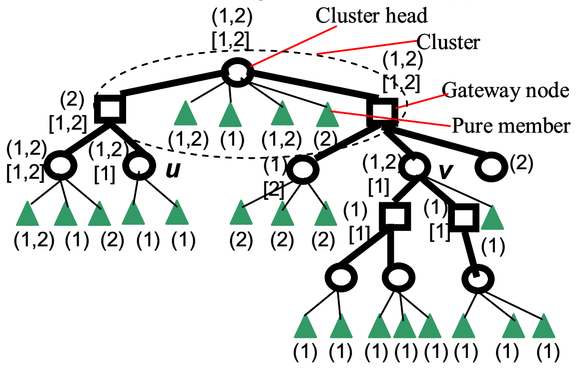

Fig. 4 M$CNet$(G) and MBT(G): at each node the upper is the group-list and the lower is the relay-list.

similar to *CNet*(G). *Group-lists* and *relay-lists* need to be updated when topology of G changes. We will discuss this in Section 5. A collision-free-multicast for a specified group can be done by using Algorithm 2. The only difference is

that in the multicast algorithm, the group ID will be transmitted with the broadcast message. When a node receives the message with the group ID, it will transmit the message to its children if the group ID belongs to its relay-list; otherwise, it will stop the transmission. For example, in Fig. 4, in a multicast for group 2, nodes u and v will stop transmission when they receive the source message with group ID 2 since 2 does not belong to their relay-lists; however, other internal nodes will transmit the message since 2 belongs to their relay-lists. The subtrees which has no node belonging to the specified group will be excluded from the multicast. In this way, a multicast will be much faster than a broadcast.

4. Assignment of Transmitting Time-Slot

In this section, we show the self-assignment of *b-time-slots* and *l-time-slots* at the internal nodes of *CNet*(G). We assume that we already have *CNet*(G). In Section 5, we will show the initialization and reconfiguration of *CNet*(G). The algorithm is incremental. Let graph $G_{old} = (V, E)$ have n ($n \geq 1$) nodes, *CNet*(G_{old}) $= (V, E_{CNet})$, and the internal nodes of *CNet*(G_{old}) have *b-time-slots* and *l-time-slots*. According to Definition 1, a graph obtained by adding a node *new* into G_{old} is a graph $G = (V \cup \{new\}, E \cup E'\}$, where $E' = \{(new, u) \mid u \in V$, and *new* and u are in the transmission range with each other$\}$. Cluster-net of G is *CNet*(G) = $(V \cup \{new\}, E_{CNet} \cup \{(new, w)\})$, where w is the parent of *new* in *CNet*(G). After *new* is added in, some nodes of *CNet*(G) need to update their *b-time-slots* and/or *l-time-slots* so that they do not violate *Time-Slot Condition 2*.

We assume that *new* has been added into *CNet*(G) but *b-time-slots* and *l-time-slots* have not been updated yet. For better clarity, we assume that at this point each node in *CNet*(G) has the following knowledge: (i) it knows its depth in *CNet*(G) and its neighbors in G, (ii) for the internal nodes of *CNet*(G_{old}), they know their *b-time-slot* and *l-time-slot*. When we say a node knows its neighbors, we mean that it knows its neighbors' knowledge. We will show how to build a *CNet*(G) with the above knowledge in Section 5.

Let v be a node at depth i in *CNet*(G), $P(v)$ be the set of nodes at depth i -1 who connect with v by edges of G, and $C(v)$ be the set of nodes at depth (i+1) who connect with v by edges of G (Fig. 6). Node v knows $P(v)$ and $C(v)$ from the assumed knowledge it has.

Algorithm 3 ***UpdateTimeSlot*** ($CNet(G)$, *new, w*)
Node *new* is a leaf in $CNet(G)$ and therefore, it does not need *b-time-slot* and *l-time-slot*. If there is one node in P(*new*) whose *l-time-slot* is different from the *l-time-slots* of other nodes in P(*new*), then Time-slot Condition 2 holds, i.e., this node can transmit the broadcast message to *new* without collision. In this case, no node needs to change its *b-time-slot* and/or its *l-time-slot*. The algorithm completes. Otherwise, *new* selects a node w from P(*new*); then sends w a message of "updating the time-slots". When the w receives the message, it executes the following steps.
Case 1: w is an internal node of *CNet*(G_{old}) (Fig 5 (a)). In this case, w has a new leaf *new*. Node w recalculates its *l-time-slot* by procedure CalculateLTimeSlot(*CNet*(G), w).
Case 2: w is a leaf of *CNet*(G_{old}) (Fig 5(b)). In this case, w has a new leaf *new* and w itself changes from a leaf to an internal node. It means that to w's parent u, w changes from leaf to internal nodes. Therefore, w updates its *l-time-slot* by procedure *CalculateLTimeSlot*(*CNet*(G), w), and u updates its *b-time-slot* and *l-time-slots* by procedure *CalculateBTime-Slot*(*CNet*(G),u) and procedure *CalculateLTimeSlot*(*CNet*(G), u).

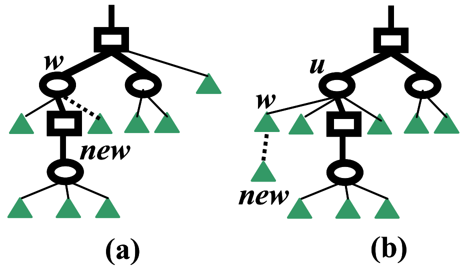

Fig. 5 Two cases of adding node *new*

The following procedure is used to calculate *b-time-slot* (*l-time-slot,* respectively) for a node y.

Procedure1
CalculateBTimeSlot(LTimeSlot, resp.) (*CNet*(G), y)
(i) Node y sends a request to ask the nodes of $C(y)$ sending their *b-time-slots* (*l-time-slots,* resp.) back in turn (Fig. 6).
(ii) When a node v in $C(y)$ receives the message, it checks the *b-time-slots* (*l-time-slots*) at the nodes of $P(v)$. If v can find two distinct *b-time-slots* (*l-time-slots*) which are different from those of the others in $P(v)$, v sends back nothing. Otherwise, v packs all different *b-time-slots* (*l-time-slots*) at the nodes of $P(v)$ and send them back to y at v's turn.
(iii) When y receives the *b-time-slots* (*l-time-slots*) from the nodes of $C(y)$, y selects the minimum positive integer which is different from all received *b-time-slots* (*l-time-slots*), and set it to be y's *b-time-slot* (*l-time-slot*).

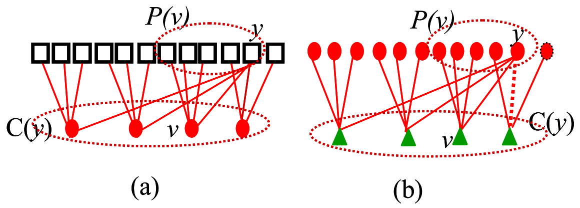

Fig. 6 (a)Calculate *b-time-slot*, and (b)Calculate *l-time-slot*

Lemma 2 (1) Procedure *CalculateBTimeSlot* (*LTimeSlot,*resp.)(*CNet*(G), y) calculates y's *b-time-slot* (*l-time-slot,* resp.) in d (D, resp.) rounds, where d is the degree of $G(V_{BT})$, and D is the degree of G; (2) in the procedure, only the knowledge at the nodes of $C(y)$ are used; and (3) the *b-time-slot* (*l-time-slot,* resp.) of y does not exceed $d(d+1)/2+1$ ($D(D+1)/2+1$, resp.).
Proof: Conclusion (2) is obvious. To prove conclusion (3), we see that in the procedure step (ii), node v in $C(y)$ sends y the time-slots at its turn only when there are no two distinct *b-time-slots* (*l-time-slots*) which are different from those of the others in $P(v)$. Therefore, v sends back at most (|$P(v)$|+1)/2 *b-time-slots* (*l-time-slots*), and y receives no

more than $\Sigma^{v}_{v \in C(y)}(|P(v)|+1)/2 \le \Sigma^{v}_{v \in C(y)}(\deg(v)+1)/2 \le$ $d(d+1)/2$ ($D(D+1)/2$) *b-time-slots* (*l-time-slots*), where deg(v) (*Deg*(v)) is the degree of v in $G(V_{BT})$ (in G). Since y selects the minimum positive integer which has to be different from all received *b-time-slots* (*l-time-slots*), the selected *b-time-slot* (*l-time-slot*) will be not larger than $d(d+1)/2$ +1 ($D(D+1)/2+1$).

Now we prove conclusion (1). It is easy to see that node y uses one round to send the request, and the nodes of $C(y)$ need $|C(y)| \le d$-1 ($\le D$-1) rounds to send the *b-time-slots* (*l-time-slots*) back to y in turn.

It is easy to show that updating the time-slots at y will only affect the nodes of $C(y)$. In order to prove the correctness, we argue that each node v of $C(y)$ can receive a broadcast message with no collision after updating: If v can find two distinct *b-time-slots* (*l-time-slots*) which are different from those of others nodes in $P(v)$, then no matter what *b-time-slot* (*l-time-slot*) that y will be assigned, v will get the broadcast message from at least one of the two distinct *b-time-slots* (*l-time-slots*). Otherwise, according to step (iii) in the procedure, the *b-time-slot* (*l-time-slot)* of y is different from all those of nodes in $P(v)$, which means that v can get the broadcast message from y without collision. □

Lemma 3 *Algorithm 3* can update *b-time-slots* and *l-time-slots* using $2d+D$ rounds in *CNet*(G) after a new node is added, and the largest *b-time-slot* (*l-time-slot,* resp.) in *CNet*(G) is not larger than $d(d+1)/2$ +1 ($D(D+1)/2+1$, respectively).

Proof: In algorithm 3, w knows that it is an internal or a leaf of *CNet*(G_{old}) from the knowledge it has. In the algorithm, at most two nodes need to recalculate their *b-time-slots* and one node needs to recalculate its *l-time-slots*. From Lemma 2, the algorithm updates the *b-time-slots* and *l-time-slots* correctly in $2d+D$ rounds, and the updated *b-time-slots* (*l-time-slots*) are not larger than $d(d+1)/2$ +1 ($D(D+1)/2+1$). □

In the proof of Lemma 2, in order to find an upper bound for *b-time-slots* (*l-time-slots*) assigned to a node y, we used inequalities $p(y) \le$ deg(y) (Deg(y)) and $C(v) \le$ deg(v) (Deg(v)). Since *deg*(x) = $C(x)$ +$P(x)$ for any node x, *b-time-slots* (*l-time-slots*) actually are around one fourth of the upper bound in Lemma 3.

5 Construction/Reconfiguration of *CNet*(*G*)

According to the definition of *CNet*(G) in Section 2 the broadcast algorithms in Section 3, and the time-slot assignment algorithms and Section 4, each node in *CNet*(G) needs to have the following knowledge:

(I) It needs to know its neighbors (it means that it need to know the neighbors' knowledge) in G and *CNet*(G), and the parent in *CNet*(G). It needs to know its status (as a cluster-head or a gateway or a pure-member).

(II) It needs to know its *b-time-slot* and *l-time-slot,* and its depth and height in *CNet(G)* if it is an internal node in *CNet(G)*. If it is the root, it knows δ (i.e., the largest *b-time-slot* in *CNet*(G)) and height of *CNet(G)*. (For multicast, nodes need to know a group list and a relay-list).

In [19], two operations, *node-move-in* and *node-move-out*, are used for constructing and reconfiguring *CNet*(G), where the nodes of *CNet*(G) maintain knowledge (I) only. In this section, we show the algorithms for maintaining knowledge (II) in both operations. There are two ways for constructing a *CNet*(G): one is to add nodes of G one by one into *CNet*(G) by using *node-move-in* operation; and the another is to do a gossip on G so that every node knows the knowledge of whole network G in O(n) rounds [7], and then each node constructs a sub-*CNet*(G) locally.

5.1 Node-Move-In Algorithm

Let graph $G_{old} = (V, E)$ have n nodes and *CNet*(G_{old}) $= (V, E_{CNet})$. A graph obtained by adding a node *new* into G_{old} is a graph $G = (V \cup \{new\}, E \cup E')$, where $E' = \{(new, u) \mid u \in V\}$ and *new* and u are in the transmission range with each other}. Cluster-net of G is defined to be *CNet*(G) = $(V \cup \{new\}, E_{CNet} \cup \{(new, w)\})$, where w is the parent of *new* in *CNet*(G). According to [19], a *node-move-in* operation can be done in $O(d_{new})$ expected rounds, where d_{new} is the degree of *new* in G. Each node in G has knowledge (I) when the operation finished.

We add two additional steps into the *node-move-in* operation of [19] as follows:

(1) Update the *b-time-slots* and *l-time-slots* by algorithm *UpdateTimeSlot*($CNet(G)$, *new, w*). Set the depth of *new* to be 1+the depth of w.

(2) Send the largest one of the updated *b-time-slots* (not more than two) back to the root. The root compares them with the existing δ, and updates δ. The nodes on the path from *new* to the root update their height as follows: *new* sends a message "updating your height" with height 0 to its parent w; when w received the message, it compares with the received height and updates its height if it needs, and then w relay the message with its height to its parent. This procedure is repeated until the root updates its height.

According to Lemma 3, step (1) can be finished in $2d+D$. Step (2) requires $2h$ rounds: h rounds for sending *b-time-slots* back to the root and another h rounds for updating the height of the nodes on the path from *new* to the root.

Theorem 2 (1) A *node-move-in* operation can be completed in $O(d_{new})$ expected rounds if the nodes of *CNet*(G) maintain knowledge (I); and it can be completed in $O(1)$ rounds if the node already know its neighbors [19]. (2) It needs additional $2h+2d+D$ rounds if the nodes maintain knowledge (II). □

5.2 Node-Move-Out Algorithm

Let graph $G_{old} = (V, E)$ have n ($n \ge 1$) nodes and *CNet*(G_{old}) $= (V, E_{CNet})$. A graph obtained by deleting a node *lev* from G_{old} is a graph $G = (V - \{lev\}, E - E')$, where $E' = \{(lev, x) \mid (lev, x) \in E\}$. We assume that G is connected. *CNet*(G_{old}) can be divided into two sub-trees: one is the tree T with *lev* as the root, and one is the tree H whose root is the root of *CNet*(G_{old}) (the case that *lev* is the root of *CNet*(G_{old}) can be dealt similarly and we will add it in the full paper) (Fig.7). Assuming that C_i (i = 1,2,3,…) are the sub-trees of *lev* in T. Since G is connected, after *lev* leaves, there exits at least one edge e in G which is neither an edge of T nor an edge of H but connects a node u of T with a node v of H.

In [19], *CNet*(G) with knowledge (I) is reconfigured in O(|T|) rounds with a *node-move-out* operation in which the nodes of T are moved into H one by one. The operation consists of the following two steps:

(Step 1) Before *lev* leaves from G_{old}, it calls *Eulerian*(*lev, T,* *message*1) in which the message1 is "finding an edge of *G* which not belong to *T*" to find the edge $e = (u, v)$ (Since *e* does not belong to *T*, *u* and *v* belongs to different sets of *T* and *H*. We suppose *u* belongs to *T*).

(Step 2) node *u* calls *Eulerian*(*u, T, message*2) in which the message is "moving the current node into *H*" to start a *Eulerian* tour in *T* from node *u* until all the nodes of *T* moved into *H*.

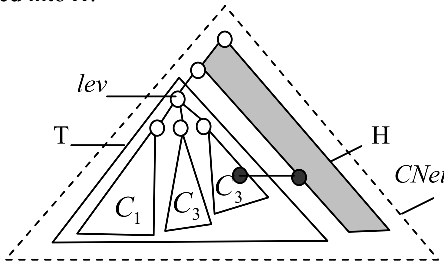

Fig. 7 Separating CNet(G) into two subtrees T and H

In our *node-move-out* operation, we need to maintain knowledge (II). Before the nodes of *T* are moved into *H*, the nodes of *H* need to delete the nodes of *T* from their neighbor lists and recalculate their *b-time-slots* and *l-time-slots* if necessary. According to Lemma 2 (2), for any node *v*, only the nodes in *P*(*v*) will use *v* to calculate their time-slots.

Our node-move-out operation has one step before and one step after the *node-move-out* operation in [19]:

(Step 0) (i) *lev* sends message "I will leave" with its height back to the root. Each node on the path from *lev* to the root updates its height according to the height it received, and then sends the message with its updated height to the next node. (ii) *lev* starts an *Eulerian* tour on *T* from *lev*. The processing at each node *x* in the tour is as follows: *x* sends out a message "delete me and recalculate time-slots" with its ID. When the neighbors of *x* in *H* received the message: (i) they delete *x* from their neighbor lists, (ii) the nodes of *P*(*x*) in *H* recalculate their *b-time-slots* and *l-time-slots* in turn, and (iii) the recalculated *b-time-slots* are sent back to *x* in turn. The largest recalculated *b-time-slot* will be kept and sends to the next node in the tour.

(Step 1 and Step 2): The steps are the same as Step 1 and Step 2 of *node-move-out* operation in [19]. However, the nodes of *T* are moved into *H* by using our *node-move-in* algorithm in Section 5.1. In our *node-move-in* operation, the largest recalculated *b-time-slot* needs to be sent back to the root. In order to save time, it is not sent back to the root, but sent to the next node in the tour so that the largest updated *b-time-slot* so far is kept in the tour.

(Step 3) The largest revised *b-time-slot* obtained in Step 0 and Step 1 & 2 are sent back to the root of *CNet*(*G*). Based on it, the root updates δ .

Theorem 3 Given a graph *G* and *CNet*(*G*), a node-move-out operation can be completed in $O(h+|T|D^2)$ rounds.

Proof: In Step 0 (ii), each node *x* in the tour invokes the nodes in *P*(*x*) to update their time-slots, which needs totally at most $|T|\cdot|P(x)|\cdot(2d+D) \leq D(2d+D)|T|$ rounds. Since our node-move-in operation needs $O(2d+D)$ rounds, Step 1 & 2 needs $O(|T|(2d+D))$ rounds. Other parts need at most $O(|T|+h)$ rounds. □

6. Simulation Results

To evaluate the average performance of the protocols, we tested them on a cluster-based WSN with the scales of 8×8 units, 10×10 units and 12×12 units, where each unit is 100 meters. The communication range of a node is 50 meters. The number of nodes used for testing varies from 64 to 720. We show the results on 10×10 units only because of the space limitation.

We tested the time (Fig. 8) and energy (Fig. 9) needed in our collision-free-flooding (CFF) broadcast protocol comparing with the results of depth-first-order (DFO) broadcast in [19]. We also tested and compared the average size and height of the backbone for a WSN (Fig. 10), and the average size of *D* and *d*, the largest degrees of the WSN and the graph induced from its backbone, respectively (Fig.11). More simulation results will be added into the full paper.

From Fig. 8 and Fig. 9 we can see that our CFF broadcast protocol is much faster and much more energy saving than the DFO broadcast protocol. Also we can see that as what we discussed at the end of Section 4, the real performance of the network is much better than the theoretical upper bound in the Lemma 3. In Lemma 3, δ and Δ are proved not larger than $d(d+1)/2$ +1 and $D(D+1)/2+1$, respectively; however, in the simulation, we found that δ and Δ are even smaller than *d* and *D*, respectively. From Fig. 10, we can see that the height of the backbone is much smaller than the size of the backbone. From Fig. 11, we note that *d* is much small that *D*. It is clear that *d* and *h* increase slowly when the number of the nodes a sensor network grows. It means that our broadcast protocol performs better when the sensor network gets denser.

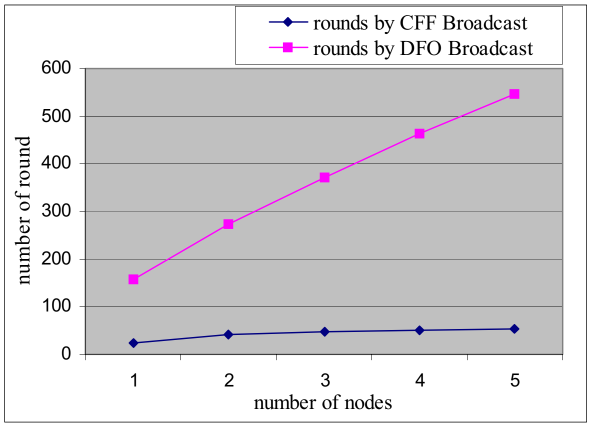

Fig. 8 Time (number of rounds) needed for completing a CFF broadcast and DFO broadcast

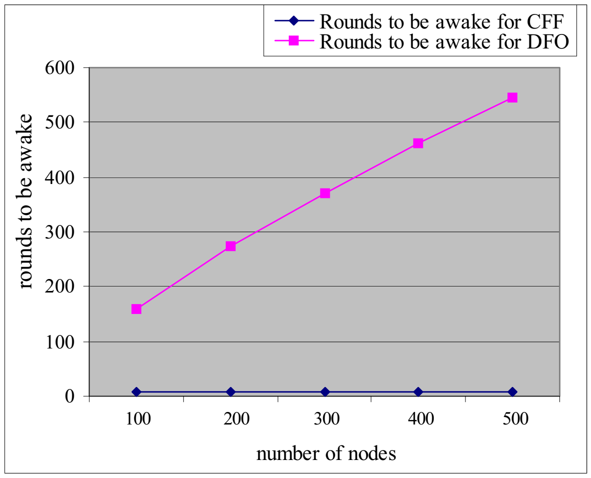

Fig. 9 Number of rounds a node needs to be awake in a CFF broadcast and a DFO broadcast

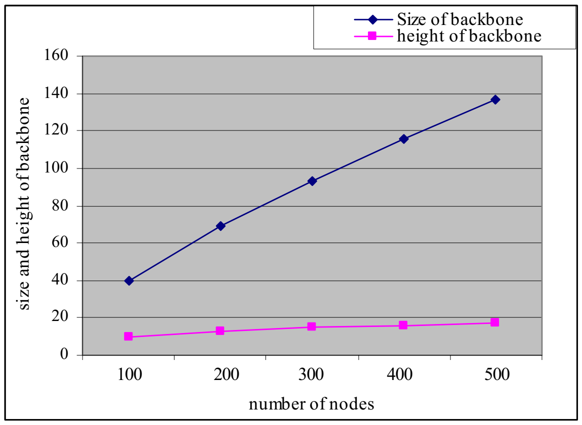

Fig. 10 Size and height of the backbone of a WSN

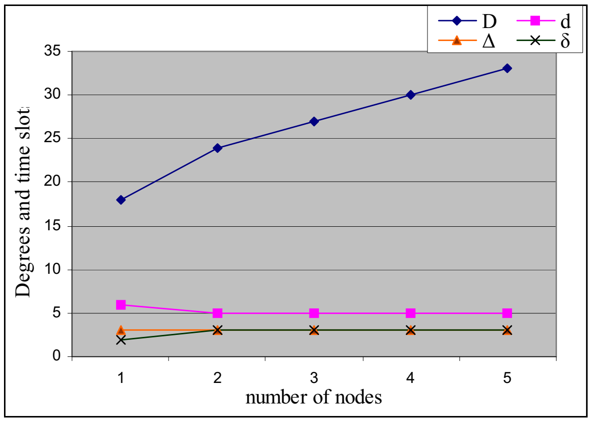

Fig.11 *D, d,* Δ and δ : maximum degree of the leaves of *CNet*(*G*) in *G*, maximum degree of the internal nodes of *CNet*(*G*) in *G*, largest *l*-time-slots assigned to the leaves of *CNet*(*G*), and largest *d*-time-slots assigned to the internal nodes of *CNet*(*G*)..

References

1. N. Alon, A. Bar-Noy, N. Linial, D. Peleg, "A lower bound for radio broadcast", Journal of Computer and System Science, no 43 (2), pp. 290-298, 1991.
2. L. Bao, J. J. Garcial-Luna-Aceves, "A new approach to channel access scheduling for ad hoc networks," Proceedings of the 7th Annual International Conference on Mobile Computing and Networking," pp. 210-221, 2001.
3. R. Bar-Ychuda, O. Goldreich, A. Itai, "On the time-complexity of broadcast in radio networks: an exponential gap between determinism and randomization, *Journal of Computer and System Science,* no 45, pp. 104-126, 1992.
4. S. Basagni, "Distributed clustering for ad hoc networks", *Proceedings of the 1999 International Symposium on Parallel Architectures, Algorithms, and Networks,* pp. 310-315, 1999.
5. S. Basagni, M. Mastrogiovanni, C. Petrioli, "A performance comparison of protocols for clustering and backbone formation in large scale ad hoc networks", *Proceedings of the 1st International Conferences on Mobile Ad-hoc and Sensor Systems,* pp. 70-79, 2005.
6. J. Blum, M. Ding, A. Thaeler, X. Cheng, " Connected dominating set in sensor networks and MANETs," Handbook of Combinatorial Optimization, Kluwer Academic Publisher, pp.329-369, 2004.
7. I. Chlamtac, A. Farago, " A new approach to the design and analysis of peer-to-peer mobile networks", *Wireless Networks,* vol. 5, no. 3, pp. 149-156, 1999.
8. I. Chlamtac, S. Kutten, "The wave expansion approach to broadcasting in multihop radio networks", IEEE Transaction on Communication, no 39(9), pp. 426-433, 1991.
9. B. S. Chlebus, L. Gasieniec, A. M. Gibbons, A. Pelc, and W. Rytter, "Deterministic broadcasting in ad hoc radio networks", *Distributed Computing* 15, pp. 27-38, 2002.
10. T. Clausen, C. Adjih, A. Laouiti, P. Minet, P. Muhlethaler, A. Qayyum, L. Viennot, "Optimized link state routing protocols," RFC 3626 Network Working Group, 2003.
11. D. Dubhashi, A. Mei, A. Panconesi, J. Radhakrishnan, A. Srinivasan, "Fast distributed algorithms for (weakly) connected dominating sets and linear-size skeletons", *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 717-724, 2003.
12. I. Gaber and Y. Mansour, "Centralized broadcast in multihop radio network", Journal of Algorithms, 46, pp. 1-20, 2003.
13. F. Kuhn, T. Moscibroda, T. Wattenhofer, "Initializing newly deployed ad hoc sensor networks", *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking,* 2004.
14. T. Moscibroda, T. Wattenhofer, "Efficient computation of maximal independent sets in unstructured multi-hop radio networks", *Proceedings of the 1st International Conferences on Mobile Ad-hoc and Sensor Systems,* pp. 70-79, 2005.
15. K. Nakano, S. Olariu, "Randomized initialization protocols for radio networks", *Handbook of Wireless Networks and Mobile Computing*, pp. 195-218, 2002.
16. S. Y. Ni, Y. C. Chen, J. P. Sheu, "The broadcast storm problem in a mobile ad hoc network," Proceedings of the 5th ACM/IEEE International Conference on Mobile Computing and Networking, pp. 151-162, 1999.
17. S. PalChaudhuri, R. Kumar, R. G. Baraniuk, "Desing of adaptive overlays for multi-scale communication in sensor networks," *DCOSS, LNCS* 3560, pp. 173-190, 2005.
18. V. Rajendran, K. Obraczka, J. J. Garcial-Luna-Aceves, "Energy-efficient collision-free medium access control for wireless sensor networks," Proceedings of the First International Conference on Embedded Networked Sensor Systems, pp. 181-192, 2003.
19. J. Uchida, I. Muzahidul, Y. Katayama, W. Chen, K. Wada, "Construction and Maintenance of A Cluster-based Architecture for Dynamic Radio Networks," 39th Hawaii International Conference on System Sciences, 2006.
20. P. J. Wan, K. M. Alzoubi, O. Frieder, "Distributed construction of connected dominating sets in wireless ad hoc networks", *ACM/Kluwer Mobile Networks and Applications, MONET,* vol. 9, no.2, pp. 141-149, 2004.
21. J. Wu, F. Dai, "Broadcasting in ad hoc networks based on self-pruning," Proceeding of Infocom'03, 2003.
22. J. Wu, H. Li, " On calculating connected dominating set for efficient routing in ad hoc wireless networks", *Telecommunication Systems,* vol. 18, no. 1/3, pp. 13-36, 2001.
23. http://www.stetson.edu/~efriedma/cirincir/