Message-Optimal Connected Dominating Sets in Mobile Ad Hoc Networks

Khaled M. Alzoubi
Department of Computer Science
Illinois Institute of Technology
Chicago, IL 60616
alzoubi@cs.iit.edu

Peng-Jun Wan
Department of Computer Science
Illinois Institute of Technology
Chicago, IL 60616
wan@cs.iit.edu

Ophir Frieder
Department of Computer Science
Illinois Institute of Technology
Chicago, IL 60616
ophir@cs.iit.edu

ABSTRACT

A connected dominating set (CDS) for a graph $G(V, E)$ is a subset V' of V, such that each node in $V - V'$ is adjacent to some node in V', and V' induces a connected subgraph. A CDS has been proposed as a virtual backbone for routing in wireless ad hoc networks. However, it is NP-hard to find a minimum connected dominating set (MCDS). Approximation algorithms for MCDS have been proposed in the literature. Most of these algorithms suffer from a very poor approximation ratio, and from high time complexity and message complexity. Recently, new distributed heuristics for constructing a CDS were developed, with constant approximation ratio of 8. These new heuristics are based on a construction of a spanning tree, which makes it very costly in terms of communication overhead to maintain the CDS in the case of mobility and topology changes.

In this paper, we propose the first distributed approximation algorithm to construct a MCDS for the unit-disk-graph with a *constant* approximation ratio, and *linear* time and *linear* message complexity. This algorithm is fully localized, and does not depend on the spanning tree. Thus, the maintenance of the CDS after changes of topology guarantees the maintenance of the same approximation ratio. In this algorithm each node requires knowledge of its *single*-hop neighbors, and only a constant number of *two*-hop and *three*-hop neighbors. The message length is $O(\log n)$ bits.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Wireless communication; F.2.2 [**Nonnumerical Algorithms and Problems**]: Geometrical problems and computations

General Terms

Algorithms, Performance

MOBIHOC'02, June 9-11, 2002, EPFL Lausanne, Switzerland.

Keywords

ad hoc networks, connected dominating set, maximal independent set, mobility.

1. INTRODUCTION

Wireless ad hoc networks can be flexibly and quickly deployed for many applications such as automated battlefield operations, search and rescue, and disaster relief. Unlike wired networks or cellular networks, no physical backbone infrastructure is installed in wireless ad hoc networks. A communication session is achieved either through a single-hop radio transmission if the communication parties are close enough (neighbors), or through relaying by intermediate nodes otherwise. In this paper, we assume that each node has a unique ID, and each node knows the IDs of all its neighbors. Scheduling of transmission is the responsibility of the MAC layer. We further assume that all nodes V in a wireless ad hoc network are distributed in a two-dimensional plane and have an equal maximum transmission range of one unit. The topology of a wireless ad hoc network can be modeled as a *unit-disk graph* [3] $G = (V, E)$, a geometric graph in which there is an edge between two nodes if and only if their distance is at most one (see Figure 1).

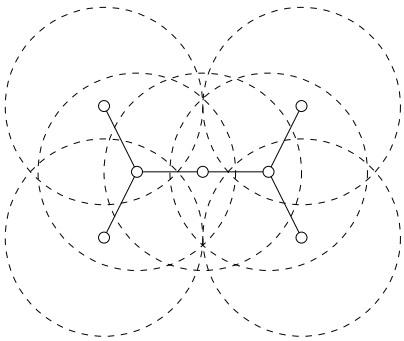

Figure 1: Modeling the topology of wireless ad hoc networks by unit-disk graphs.

Although a wireless ad hoc network has no *physical* backbone infrastructure, a *virtual* backbone can be formed by

nodes in a connected dominating set (CDS) of the corresponding unit-disk graph [2] [4][5]. In general, a *dominating set* (DS) of a graph $G = (V, E)$ is a subset $V' \subset V$ such that each node in $V - V'$ is adjacent to some node in V', and a CDS is a dominating set that also induces a connected subgraph. A (connected) dominating set of a wireless ad hoc network is a (connected) dominating set of the corresponding unit-disk graph. A virtual backbone, also referred to as a *spine*, plays a very important role in routing, where the number of nodes responsible for routing can be reduced to the number of nodes in the CDS. The virtual backbone also plays an important role for broadcasting and connectivity management in wireless ad hoc networks [2]. Broadcasting responsibility can be reduced to the nodes in the CDS instead of all the nodes in the graph. To reduce the communication overhead, to increase the convergence speed, and to simplify the connectivity management, it is desirable to find a CDS of a small number of nodes.

The *quality* of a virtual backbone in wireless ad hoc networks is conventionally measured by its approximation factor, which is the ratio of its size to that of the MCDS; and the *construction cost* of a virtual backbone is measured by the message complexity and the time complexity. Table 1 [7] summarizes the quality and construction costs of the virtual backbones proposed in [5], [8], [6] and [7]. In *mobile* wireless ad hoc networks where nodes may move continuously, the virtual backbone should not only preserve good quality *all the time* instead of at a particular moment, but also allow efficient maintenance due to topology changes. Unfortunately, none of the virtual backbones proposed in the literature can be efficiently maintained to preserve good quality. The virtual backbone proposed in [5][4][2] not only has poor quality, but is also very expensive to be maintained as it requires the maintenance of a spanning tree. The virtual backbones proposed in [8] and [6] are relatively easier to be maintained, but suffer from poor quality. On the other hand, the virtual backbone proposed in [1][7] is of good quality, but is very hard to be maintained as its good quality relies on a ranking of nodes depending on a rooted spanning tree.

Backbone	[5][4][2]	[8]	[6]	[1][7]
Quality	$\Theta(\log n)$	n	$\frac{n}{2}, n$	≤ 8
Messages	$O(n^2)$	$\Theta(m)$	$O(n^2)$	$O(n \log n)$
Time	$O(n^2)$	$O(\Delta^3)$	$\Omega(n)$	$O(n)$

Table 1: Summary of performance of various virtual backbones: n is the number of nodes, m is the number of edges, and Δ is the maximum nodal degree.

The heuristics proposed by Alzoubi et al in [1][7] take advantage of the property of the maximal independent set (MIS), where the approximation ratio for any heuristic for constructing a MIS in a unit-disk graph is at most 5. An MIS S is an independent DS, i.e. all pairwise nodes in S are non-adjacent. Thus the construction of a MIS is a construction of an DS with approximation ratio of 5. Also the maintenance of the DS, as topology changes can be described as the maintenance of a MIS for the unit-disk graph.

In this paper we take advantage of the MIS properties to construct a CDS for wireless ad hoc networks, that is easy to maintain under the environment of mobility and topology changes. The construction of the CDS consists of two *localized* phases: the first phase is the construction of the MIS. MIS nodes are referred to as dominators. In the second phase each dominator is connected to all dominators within *three*-hop distance. The connecting nodes between any pair of dominators that are at most *three*-hop away from each other are referred to as connectors. Both of the dominator and connector nodes form the CDS. The key technique in our maintenance approach is to maintain the MIS in the unit-disk graph, and to maintain connectivity between all MIS nodes. Every time a new dominator appears in a new vicinity, it must have a connection through connector nodes to all dominators within *three*-hop distance. A connector node maintains its state as a connector as long as it connects at least two dominators. Otherwise it changes its state to dominatee if it has at least one dominator. In this paper, we show that the number of dominators within *three*-hop distance from a dominator is constant.

Our algorithm makes the following two main contributions to the virtual backbone in mobile ad hoc networks:

1. The virtual backbone proposed in this paper not only has a constant approximation factor, but also can be constructed in both linear messages and linear time. Thus, it is the first virtual backbone with all those properties.

2. The virtual backbone can be maintained efficiently such that its good quality can be preserved all the time. The updating of the virtual backbone occurs only at a topology change. In addition, each updating is confined locally to a small neighborhood.

The remainder of this paper is organized as follows. In section 2, we introduce the structure of the virtual backbone and study its quality. In Section 3, we present a distributed construction of this virtual backbone with linear communication cost and computation cost. In Section 4, we describe a local maintenance of this virtual backbone in the mobile environment. Finally, we conclude this paper in section 5.

2. VIRTUAL BACKBONE

Our virtual backbone U is a union of two subsets of nodes S and C. The nodes in S form a maximal independent set (MIS), i.e., an independent dominating set. Note that any pair of nodes in S are separated by at least two hops, and any subset of nodes in S is at most three hops away from the rest nodes in S. The nodes in C are then selected as follows: for each pair of nodes in S at most three hops away from each other, choose a path of at most three hops (not necessarily the shortest one) between them, and add the internal nodes in the chosen path to C. Obviously, U is a CDS. In the rest of this section, we show that U is within constant factor of the MCDS.

We begin with some general geometric properties of a MIS S in a unit-disk graph. The following lemma bounds the size of any MIS. It is well-known that each node is adjacent to at most five nodes in S. The following lemma gives constant bounds on the number of nodes in S that are nearby any node in S.

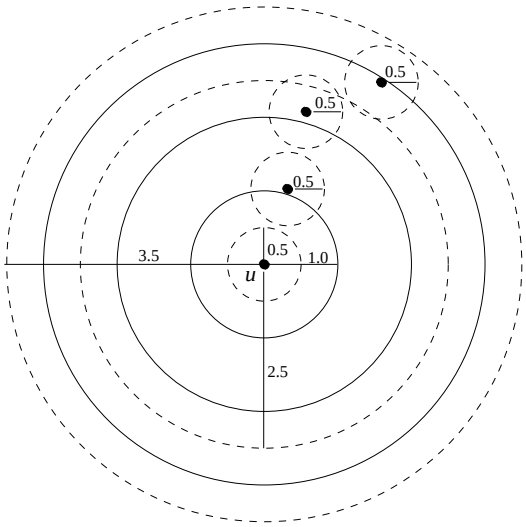

Figure 2: The disks of radius 0.5 centered at the nodes in S that are within three hops away from u all lie within the annulus centered at u of radii 0.5 and 2.5 and are disjoint.

LEMMA 1. *Let S be any MIS of the unit-disk graph G and u be an arbitrary node in S.*

1. *The number of nodes in S that are exactly two hops away from u is at most 23.*
2. *The number of nodes in S that are at most three hops away from u is at most 47.*

PROOF. The proof following from the standard area argument. The disks of radius 0.5 centered at the nodes in S that are exactly two hops away from u all lie within the annulus centered at u of radii 0.5 and 2.5 and are disjoint. Thus, the number of nodes in S that are exactly two hops away from u is less than

$$\frac{\pi \cdot 2.5^2 - \pi \cdot 0.5^2}{\pi \cdot 0.5^2} = 24.$$

Similarly, the disks of radius 0.5 centered at nodes in S that are at most three hops away from u all lie within the annulus centered at u of radii 0.5 and 3.5 and are disjoint (see Figure 2). So the number of nodes in S that are exactly three hops away from u is less than

$$\frac{\pi \cdot 3.5^2 - \pi \cdot 0.5^2}{\pi \cdot 0.5^2} = 48.$$

□

Let *opt* denote the size of a minimum CDS in G. The following lemma bounds the size of any MIS in G and was proven in [7].

LEMMA 2. *The size of any independent set in G is at most $4 \cdot opt + 1$.*

Based on the above two lemmas, we can prove the constant-approximation of U to the MCDS.

THEOREM 3. *The size of CDS U is within a constant factor of opt.*

PROOF. From Lemma 1, the total number of pairs of nodes in S that are within three hops away from each other is at most $\frac{47|S|}{2}$. Since each of such pairs introduces at most two nodes to C, the total number of nodes in C is at most $47\,|S|$. Thus, the total number of nodes in U is at most $48\,|S|$. From Lemma 2, the total number of nodes in the CDS U is at most

$$|U| \leq 48\,|S| \leq 192opt + 48.$$

Therefore, the size of CDS U is within a constant factor of *opt*. □

3. DISTRIBUTED CONSTRUCTION

By definition, any pair of nodes in a MIS are separated by at least two hops. However, a subset of nodes in a MIS U may be three hops away from its complementary subset in U. This case may appear when an ID-Based approach is used for rank assignment [1]. Our distributed construction of the CDS can be briefly described as two phases. The first phase, a MIS S is constructed. The nodes in S are referred to as *dominators*, and the nodes not in S are referred to as *dominatees*. In the second phase each dominatee node identifies the dominators that are at most two hops away from itself and broadcasts this information. Using such information from all neighbors, each dominator node identifies a path of at most three hops (not necessarily the shortest one) to each dominator that is at most three hops away from itself and has *larger* ID than its own ID, and informs all nodes in this path about this selection. The set C then consists of all dominatee nodes in these paths, which are referred to as *connectors*. However, the description of our CDS construction combines the two phases. In the next, we describe a distributed algorithm with *linear* message complexity and *linear* time complexity to implement this distributed construction.

3.1 Local Variables and Structures

Each node is in one of the four states: candidate, dominatee, dominator and connector. Each node is initially in the candidate state and subsequently enters either the dominatee state or the dominator state. The connector state can only be entered from the dominatee state.

Each node also maintains several local variables and data structures. The local variable x_1 stores the number of current candidate neighbors, and is initially equal to the total number of neighbors. The local variable x_2 stores the number of current candidate neighbors with lower IDs, and is initially equal to the total number of neighbors with lower IDs. Note that both x_1 and x_2 can be initialized in linear time.

Each dominatee or connector node maintains a local variable y which counts the number of neighboring dominatees that have reported their list of adjacent dominators. y initially equals to 0.

Each dominator node maintains a local variable z which counts the number of reports yet to be received from its neighbors on their lists of single-hop dominators and lists of two-hop dominators. z initially equals to *twice* the number of neighbors.

Each dominatee node maintains two lists, $list_1$ and $list_2$. $list_1$ stores the IDs of the neighboring dominators. Each entry in $list_1$ is simply the ID of neighboring dominator. $list_2$ stores the IDs of the dominators two hops away and the IDs of the neighboring dominatee to reach these dominators. Each entry in $list_2$ is an ordered pair of the ID of a dominator two hops away and a neighboring dominatee that is adjacent to both. All entries in both lists are sorted in the increasing order of the IDs of the dominators, and both lists initially are empty.

Each dominator node maintains two lists, $list_2$ and $list_3$. $list_2$ (respectively, $list_3$) stores the ID of the dominators with *larger* IDs that are two (respectively, three) hops away and the IDs of its neighbors to reach these dominators. An entry in $list_2$ (respectively, $list_3$) is an ordered pair of the IDs of a dominator with larger ID that is two (respectively, three) hops away and a neighbor to reach this dominator. All entries in both lists are sorted in the increasing order of the IDs of the dominators, and all lists initially are empty.

Each connector node maintains a list *Rlist* which is initially empty. Each entry in *Rlist* contains two parameters. The first parameter is a pair of IDs of two dominators to which it maintains connectivity. The second parameter contains the ID of the associated connector that connects the two dominators in the first parameter, if the two dominators are three hop distance. If the two dominators are two hop distance, the value of the second parameter is assigned to Null.

Each node further maintains a list *Clist* which is initially empty and stores the IDs of neighboring connectors.

3.2 Messages and Actions

A candidate node with $x_2 = 0$ changes its own state to dominator, initializes z to *twice* the number of neighbors, and then broadcasts a DOMINATOR message. Note that such node does exist at the beginning.

Upon receiving a DOMINATOR message, a node (which cannot be a dominator node) decrements x_1 by one and inserts the ID of the sender into $list_1$. A candidate node further proceeds as follows. It changes its own state to dominatee, and then broadcasts a DOMINATEE message. If $x_1 = 0$ after the updating, it broadcasts a LIST1 message which contains all entries in $list_1$; if the number of neighboring dominators is also equal to the number of neighbors (i.e., all neighbors are dominators), it also broadcasts a LIST2 message which contains all entries in $list_2$ (which is empty in this case).

Upon receiving a DOMINATEE message, a candidate node decrements x_1 by one. If the sender has lower ID, it decrements x_2 by one. If $x_2 = 0$ after the updating, it first changes its own state to dominator, then initializes z to *twice* the number of neighbors, and finally broadcasts a DOMINATOR message.

Upon receiving a DOMINATEE message, a dominatee node decrements x_1 by one. If $x_1 = 0$ after the updating, it broadcasts a LIST1 message which contains all entries in $list_1$.

Upon receiving a LIST1 message, a dominatee or connector node increments y by one. (When a node receives a LIST1 message, the node cannot be in candidate state. However, some of its neighbors may be still in the candidate state and thus it can not determine the final number of neighboring dominatees. This is why we increment y.) For each dominator ID contained in the LIST1 message which does not appear in the current $list_1$ and $list_2$, it inserts into $list_2$ an entry consisting of this dominator ID and the sender's ID. Finally, if $x_1 = 0$ and y is also equal to the number of neighbors minus the number of neighboring dominators (the length of $list_1$) after the updating, it broadcasts a LIST2 message which contains all entries in $list_2$.

Upon receiving a LIST1 message, a dominator node decrements z by one. For each dominator ID contained in the LIST1 message which is larger than its own ID and does not appear in the current $list_2$, it inserts into $list_2$ an entry consisting of this dominator ID and the sender's ID, and removes from $list_3$ the entry containing this dominator ID if there is any. If $z = 0$ after the updating, it broadcasts a LIST3 message which contains all entries in $list_2$ and $list_3$.

Upon receiving a LIST2 message, a dominator node decrements z by one. For each dominator ID contained in the LIST2 message which is larger than its own ID and does not appear in the current $list_2$ and $list_3$, it inserts into $list_3$ an entry consisting of this dominator ID and the sender's ID. If $z = 0$ after the updating, it broadcasts a LIST3 message which contains all entries in $list_2$ and $list_3$.

Upon receiving a LIST3 message, a node (which must be either in dominatee state or in connector state) checks whether its ID appears in any of the entries in this message, and if so it proceeds as follows. First, it sets its state to connector if its current state is dominatee. Then, for each entry in LIST3 message that has its ID, it inserts into the first parameter of its *Rlist* the ID of the sender, and the ID of the dominator it is responsible to connect (target dominator). If the target dominator is adjacent to itself, it sets the second parameter to null. Otherwise, it sets the second parameter to the ID of neighboring node associated with the target dominator in its own $list_2$. Finally, it broadcasts a CONNECTOR1 message which includes two parameters, the first parameter has its own ID, and the second parameter contains a list of all entries that were added to its *Rlist*.

Upon receiving a CONNECTOR1 message, a node inserts into its *Clist* the ID of the sender. A node which is not a dominator further checks whether its ID appears in any of the entries of the second parameter of the message, and if

so it proceeds as follows. First, it sets its state to connector if its current state is dominatee. Then, it inserts into the first parameter of its *Rlist* the first parameter of the entry that has its ID in the received CONNECTOR1 message, and adds the ID of the sender to the second parameter in its *Rlist*. Finally, it broadcasts a CONNECTOR2 message.

Upon receiving a CONNECTOR2 message, a node inserts into its *Clist* the ID of the sender.

Figure 3 illustrates the construction process of the CDS. In the graph, the IDs of the nodes are labelled beside the nodes. White nodes represent the candidate nodes, black nodes represent the dominators, gray nodes represent the dominatees, and the white node with an inner black node represents a connector node. A possible execution scenario is shown in Figure 3(a)–(d), which is explained below.

1. Initially all nodes are candidates (see Figure 3(a)).
2. Each of the nodes 1, 2, 3 and 4 declares itself as a dominator, and broadcasts a DOMINATOR message. Notice this process may occur simultaneously, since each one of these nodes has the lowest ID among all its neighbors. Whenever a neighboring node receives the DOMINATOR message, it declares itself as a dominatee and broadcasts a DOMINATEE message. Thus each of the nodes 5, 6 and 7 declares itself as a DOMINATEE and broadcasts a DOMINATEE message. (see Figure 3(b)).
3. Upon receiving DOMINATOR and DOMINATEE messages from all its neighbors; node 5 sends a LIST1 message, which includes the IDs of nodes 1 and 2; node 6 sends a LIST1 message, which includes the IDs of nodes 3 and 4; and node 7 sends a LIST1 message, which includes the IDs of nodes 3 and 4.
4. Upon receiving the LIST1 message from node 5, node 6 sends a LIST2 message, which includes the IDs of nodes 1 and 2. Upon receiving the LIST1 message from node 6, node 5 sends a LIST2 message, which includes the IDs of nodes 3 and 4. Since all neighbors of node 7 are dominators, node 7 sends a LIST2 message with the empty list $list_2$.
5. Upon receiving LIST1 and LIST2 messages from node 5, node 1 selects node 5 as a connector to reach nodes 2, 3 and 4 by sending a LIST3 message. Upon receiving LIST1 and LIST2 messages from node 5, node 2 selects node 5 as a connector to reach nodes 3 and 4 by sending a LIST3 message. Upon receiving LIST1 and LIST2 messages from nodes 6 and 7, node 3 selects node 7 as a connector to reach node 4 by sending a LIST3 message. Notice, node 4 does not make any selection since it has the largest ID among all dominators within *three*-hop distance.
6. Upon receiving LIST3 message from nodes 1 and 2, node 5 declares itself as a connector for each of the pairs (1,2), (1,3), (1,4), (2,3) and (2,4), then it sends a CONNECTOR1 message selecting node 6 as a second connector to connect each of the nodes 1 and 2 to each of the nodes 3 and 4. Upon receiving LIST3 message from node 3, node 7 declares itself as a connector for the pair (3,4), then it sends a CONNECTOR1 message. (see Figure 3(c)).
7. Upon receiving the CONNECTOR1 message from node 5, nodes 6 declares itself as a connector for each of the pairs (1,3), (1,4), (2,3) and (2,4) and it broadcasts CONNECTOR2 message.(see Figure 3(d)).

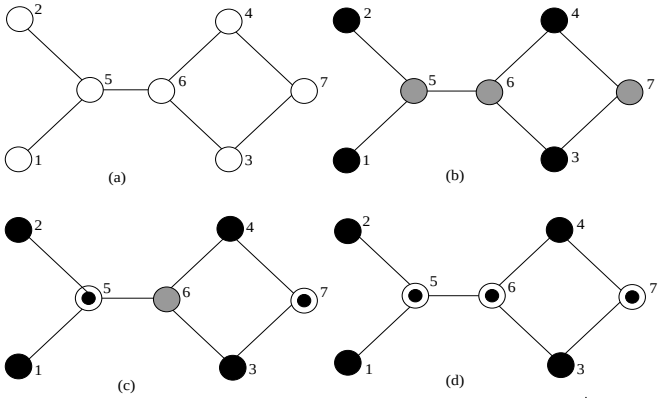

Figure 3: Example: CDS Construction

3.3 Message and Time Complexity

THEOREM 4. *Our distributed algorithm for constructing a CDS has an $O(n)$ time complexity, and $O(n)$ message complexity.*

PROOF. The worst case time complexity for the MIS occurs when all nodes are arranged in either ascending or descending order and the maximum nodal degree is 2. In this case each node has to wait for all other nodes with lower ids. Assume we have the graph with the n nodes $(v_1, v_2, ..., v_n)$, then each node v_i must wait for its neighbor node v_{i-1} to declare its state. Each node must wait one time unit more than the waiting time of the previous node. Node v_n has to wait the longest $(n-1$ units). Also each node sends only one message either a dominator or dominatee message. In connecting the MIS, each dominatee node waits $O(\Delta)$ time to build its $list_1$ and $list_2$. A dominator node waits $O(\Delta)$ time for LIST1 and LIST2 messages from all its neighbors before it selects a connector. The rest of procedures have a constant time. Since each node sends a constant number of messages, the total number of messages is $O(n)$. Thus, both the time and message complexity of our algorithm is $O(n)$. □

4. MOBILE MAINTENANCE

We need to maintain a connected dominating set in the unit-disk graph as the topology of the network may change. In the mean time we need to maintain the same performance ratio for the CDS. The key technique in our approach is to maintain the MIS in the unit disk graph first, and to maintain the connection between all MIS nodes within *three*-hop

distance through connector nodes. In our discussion for the maintenance of the CDS, we need to distinguish between dominators and connectors. After any topology changes, the MIS should be maintained, but there may be an additional affect on the connectors. When a dominator node is turned off, or leaves its vicinity, this changes should affect the connectors, which are used *only* to connect this dominator to other dominators. After the MIS is maintained and the connectors are changed back to dominatees whenever is needed, the next step is to make sure that any dominator appears in a new vicinity must have a *two*-hop and *three*-hop path of connector nodes to all *two*-hop and *three*-hop dominators respectively. In the next, we provide a brief description of the maintenance process. The implementation details of this process will appear in Alzoubi's Dissertation.

4.1 Dominator Node Movement

When a *dominatee* or *connector* node v learns that a dominator node u has left its vicinity and u is the only dominator of v, v changes its own state to candidate and then it sends a WARNING1 message reporting the loss of u. The WARNING1 message contains v's ID and state, and the ID of u. If v has other dominators and v is a dominatee, it simply remains as a dominatee. If v is a connector connecting two or more dominators other than u, it remains as a connector. Otherwise, it changes its state to dominatee, and sends an SDOMINATEE message.

Whenever a *dominatee* node w receives a WARNING1 message from v reporting the loss of u, it sends a RESPONSE, which contains w's ID and state, and the ID of u.

Whenever a *connector* node w receives a WARNING1 message from a dominatee node v, or from a connector node v which is not in w's *Rlist*, w maintains its state as a connector, and sends a RESPONSE message. Whenever a *connector* node w receives a WARNING1 message from a connector node v, and w is *only* responsible to connect u to other dominators, w changes its state to dominatee and sends a RESPONSE message. Otherwise, w maintains its state as a connector, and sends a RESPONSE message. Whenever a *connector* node w receives an SDOMINATEE message from a connector node v, and w is *only* responsible to connect u to other dominators, w changes its state to dominatee and sends an SDOMINATEE message. Otherwise, w maintains its state as a connector.

Whenever candidate node receives a WARNING1, or RESPONSE message from each neighbor, it applies the CDS algorithm locally. Thus, a candidate node v with the lowest ID declares itself as a dominator. Then v must be connected through connector nodes to all dominators within *three*-hop distance by applying the CDS algorithm locally.

When a dominator node u joins a neighborhood with at least one dominator, the dominator with the lowest ID becomes a *winner*, and maintains its state as a dominator. All other dominators switch their state to a dominatee. Otherwise, u (*winner*) maintains its state as a dominator, and sends a DOMINATOR message. However, the *winner* must be connected through connector nodes to all dominators within *three*-hop distance by applying the CDS algorithm locally.

4.2 Dominatee or Candidate Node Movement

When a dominatee node v joins a new neighborhood, if any of its new neighbors is a dominator, it maintains its state as a dominatee, it also sends a LIST1 message. When receiving a LIST1 message from v, a dominatee or connector node w sends a LIST1 and LIST2 messages. Whenever v receives a LIST1 message from each dominatee and connector neighbor, it sends a LIST2 message. Then the dominators receiving the LIST1 and LIST2 messages react based on the CDS algorithm.

When a dominatee node u joins a new neighborhood, if non of its new neighbors is a dominator, it declares itself as a dominator and sends a DOMINATOR message. If a DOMINATOR message is received from y, a dominatee or connector node v sends a LIST1 message, followed by LIST2 message. When receiving a LIST1 message from a dominatee or connector neighbor v, a dominatee or connector node w sends an LIST2 message. Then the dominators receiving the LIST1 and LIST2 messages react based on the CDS algorithm.

Whenever a new node y joins the network, initially it is a candidate, if any of its neighbors is a dominator, it becomes a dominatee, and the same action is taken as if a dominatee node joins a new neighborhood.

Whenever a new node y joins the network, initially it is a candidate, if non of its neighbors is a dominator, it declares itself as a dominator, and sends a DOMINATOR message. Then the same action is taken as if a dominatee node joins a new neighborhood and becomes a dominator.

4.3 Connector Node Movement

Whenever a connector node w learns that a connector node v has left its vicinity, if w is *only* responsible to connect v to other dominators, it changes its own state to a dominatee. Otherwise, it maintains its own state as a connector. However, w sends a LOST message reporting the lose of connection to the dominators (lost dominators) associated with it through the connector node v.

Whenever a dominator node x receives a LOST message from w, if any of the lost dominators has a larger ID than its own, it sends a REQUEST message, which contains its own ID, and the IDs of all lost dominators with larger IDs. Whenever a dominatee or a connector node x receives the REQUEST message, it sends a REPLY message, which contains its own ID and for each dominator appeared in the REQUEST message the ordered pair (ID, distance), where distance is, equal to 1 if the dominator is *one*-hop from x, equal to 2 if the dominator is *two*-hop from x, or equal to ∞ otherwise.

Whenever a dominator x receives a REPLY message from a neighbor, it selects new connectors for all *two*-hop and *three*-hop dominators, then the CDS continues to be applied locally.

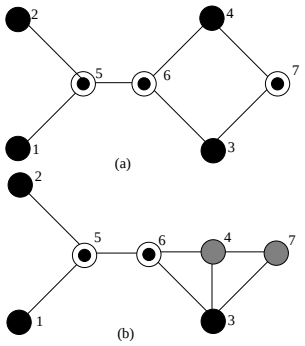

Figure 4: Example of one node movement

Whenever a dominator node u learns that a connector node v has left its vicinity, if any of the dominators (lost dominators) connected to u through v has a larger ID than its own, it sends a REQUEST message, which contains its own ID, and the IDs of all lost dominators with larger IDs. Whenever a dominatee or a connector node x receives the REQUEST message, it sends a REPLY message, which contains its own ID and for each dominator appeared in the REQUEST message the ordered pair (ID, distance), where distance is, equal to 1 if the dominator is *one*-hop from x, equal to 2 if the dominator is *two*-hop from x, or equal to ∞ otherwise.

Whenever a dominator x receives a REPLY message from a neighbor, it selects new connectors for all *two*-hop and *three*-hop dominators, then the CDS continues to be applied locally.

4.4 Examples

Figure 4(a,b) illustrates the action taking by neighboring nodes in response to a dominator node movement. Figure 4(a) represents the network topology before the node movement. When node 4 moves and becomes within the vicinity of the dominator node 3, and since it has a higher ID than node 4, it changes its state to a dominatee and sends an SDOMINATEE message. When node 7 receives the SDOMINATEE message from node 4, it removes each pair in its *Rlist* associated with the node 4. Since node 7 has only one entry in its *Rlist*, and this entry corresponds to node 4, node 7 switches to dominatee and sends an SDOMINATEE message.

Figure 5(a-e) illustrates the action taking by neighboring nodes in response to a dominator node movement and two broken links simultaneously. Figure 5(a) represents the network topology before the node movement. The execution procedures are explained below:

1. When the dominator node 1 moves upward, both of the dominatee nodes 4 and 5 become candidate nodes (see 5(b)), and both of them send a WARNING message.

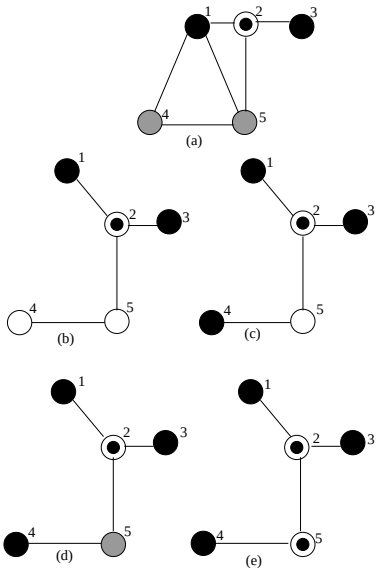

Figure 5: Dominator node movement with two broken links simultaneously.

2. Whenever node 2 receives the WARNING message from node 5, it sends back a RESPONSE message. Whenever node 4 receives the WARNING message from node 5, it declares itself as a dominator (since it has the lowest degree among all its candidate neighbors) (see 5(c)) and sends a DOMINATOR message.

3. Whenever node 5 receives the DOMINATOR message, it declares itself as a dominatee (see 5(d)) and sends a DOMINATEE message, followed by LIST1 and LIST2 messages.

4. Whenever node 2 receives the LIST1 message it sends a LIST2 message.

5. Whenever the dominator node 3 receives the LIST2 message from node 2, it sends a LIST3 message selecting node 2 as a connector to reach the dominator node 4 (since it has a lower ID than the dominator node 4).

6. Whenever node 2 receives the LIST3 message, it selects node 5 as a second connector to reach node 4, by sending a CONNECTOR1 message.

7. Whenever node 5 receives the CONNECTOR1 message addressed to itself, it declares itself as a connector (see 5(e))and sends a CONNECTOR2 message.

5. CONCLUSIONS

In this paper, we proposed the first message-optimal distributed approximation algorithm for constructing and maintaining a CDS with a constant approximation ratio in linear time and linear messages. The algorithm is fully localized, and it does not rely on the spanning tree construction, which makes it practical for situations, where the topology changes are frequent and unpredictable. We also provided

the procedures to be followed in case of mobility and topology changes. This algorithm outperforms all existing algorithms for mobile ad hoc networks, in terms of the CDS size, the message complexity, the time complexity, the message size, and the memory required to maintain neighborhood knowledge. In addition, each node only requires knowledge of its *single*-hop neighbors, and only a constant number of *two*-hop and *three*-hop neighbors, and the message length is $O(\log n)$ bits. We also showed the importance of the MIS on maintaining a low approximation ratio for the construction and maintenance of the CDS.

6. REFERENCES

[1] K. M. Alzoubi, P.-J. Wan, O. Frieder, "Distributed Heuristics for Connected Dominating Set in Wireless Ad Hoc Networks", to appear in *Journal on Communication Networks*, 2002.

[2] V. Bharghavan and B. Das, "Routing in Ad Hoc Networks Using Minimum Connected Dominating Sets", *International Conference on Communications'97*, Montreal, Canada. June 1997.

[3] B. N. Clark, C. J. Colbourn, and D. S. Johnson, "Unit Disk Graphs", *Discrete Mathematics*, 86:165–177, 1990.

[4] B. Das, R. Sivakumar, and V. Bhargavan, "Routing in Ad-Hoc Networks Using a Spine", *International Conference on Computers and Communications Networks '97*, Las Vegas, NV. September 1997.

[5] R. Sivakumar, B. Das, and V. Bharghavan, "An Improved Spine-based Infrastructure for Routing in Ad Hoc Networks", *IEEE Symposium on Computers and Communications '98*, Athens, Greece. June 1998.

[6] I. Stojmenovic, M. Seddigh, J. Zunic, "Dominating sets and neighbor elimination based broadcasting algorithms in wireless networks", *Proc. IEEE Hawaii Int. Conf. on System Sciences*, January 2001.

[7] P.-J. Wan, K. M. Alzoubi, O. Frieder, "Distributed Construction of Connected Dominating Set in Wireless Ad Hoc Networks", to appear in *IEEE INFOCOM 2002*.

[8] J. Wu and H. L. Li, "On calculating connected dominating set for efficient routing in ad hoc wireless networks", *Proceedings of the 3rd ACM international workshop on discrete algorithms and methods for mobile computing and communications*, 1999, Pages 7–14.