Tree-Based Virtual Backbone Construction in Mobile Ad Hoc Networks

by

Kazuya Sakai

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
August 9, 2010

Keywords: virtual backbone, connected dominating set, CDS, ad hoc networks

Approved by

Wei-Shinn Ku, Chair, Assistant Professor of Computer Science and Software Engineering
Alvin Lim, Associate Professor of Computer Science and Software Engineering
Xiao Qin, Assistant Professor of Computer Science and Software Engineering

Abstract

The connected dominating set (CDS) has been extensively used for routing and broadcast in mobile ad hoc networks. While existing CDS protocols are successful in constructing CDS of small size, they either require localized information beyond immediate neighbors, lack the mechanism to properly handle nodal mobility, or involve lengthy recovery procedure when CDS becomes corrupted. In this paper, we introduce the tree-based CDS protocols, which first elect a number of initiators distributively and then utilize timers to construct a CDS from initiators with the minimum localized information. We demonstrate that our CDS protocols are capable of maintaining CDS in the presence of changes of network topology. Depending on the number of initiators, there are two versions of our tree-based CDS protocols. The Single-Initiator (SI) works the best when the network diameter is given. On the other hand, the Multi-Initiator version (MI) built on top of SI is a localized CDS protocol and possesses all of the advantages of SI. We evaluate our protocols by both the ns-2 simulation and an analytical model. Compared with the other known CDS protocols, the simulation results demonstrate that both SI and MI produce and maintain CDS of very competitive size. The analytical model shows the expected convergence time and the number of messages required by SI and MI in the construction of CDS, which match closely to our simulation results. This helps to establish the validity of our simulation.

In addition to the two tree-based CDS protocols, we proposed the two post optimization techniques, Fast-Convergence Dominator Tree Construction (FC-DTC) and Extended Mobility Handling (EMH). FC-DTC significantly reduces the convergence time required by SI and MI, and EMH enables SI and MI to adapt to the topology changes more efficiently. The simulation and analytical studies demonstrate these extensions drastically improve the performance of SI and MI.

Acknowledgments

I would like to thank people who guided and supported me during my study at Auburn University. Without their contributions, this research would not be possible. I appreciate my academic advisor, Dr. Wei-Shinn Ku, for his guidance and assistant. I am also grateful to Dr. Alvin Lim and Dr. Xiao Qin for serving my committee. There were some helps from people outside of Auburn University. I would like to thank Dr. Min-Te Sun at National Central University for his help for my research. Additionally, I would like to show my great appreciation to my mother for her support during my graduate study.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Mobile ad hoc networks (MANETs) are well-suited for communications in sensor networks as well as the battlefield and rescue missions where a fixed infrastructure is not readily available [1]. The effectiveness of many communication primitives for MANETs, such as routing [2], multicast/broadcast [3], and service discovery [4], rely heavily on the availability of a virtual backbone. A virtual backbone of a wireless network is typically the connected dominating set (CDS) of the graph representation of the network. It is defined as a subset of nodes in a network such that each node in the network is either in the set or a neighbor of some nodes in the set, and the induced graph of the nodes in the set is connected.

In general, the smaller the CDS is, the less communication and storage overhead the protocols making use of CDS will incur. Hence, it is desired that the size of the CDS for MANETs to be as small as possible. On the other hand, it is known that the problem of finding the minimum CDS is NP-hard [5]. The problem becomes even more intriguing when no node has the view of the complete network topology, which is generally the case in most MANETs. As a result, the existing CDS protocols for MANETs [6–19] emphasize on constructing a small CDS distributively with localized information. While these protocols are successful in creating a small CDS, they either lack the mechanism to maintain the CDS transparently when network topology changes from time to time or incur large amount of control overhead. Notice that the topology change can be the result of node mobility, the power outage of some nodes in the network, the deployment of additional nodes to the network, or the combination of the aforementioned cases. The reconstruction of CDS from scratch can keep the CDS from being available for service for an extended period of time.

In this paper, we first introduce a tree-based Connected Dominating Set protocol, namely Single-Initiator (SI). In the first phase, SI distributively elects a unique initiator. In the second phase, SI grows a dominator tree from the initiator to form the CDS by using timers. While SI has advantage over the CDS protocols in [6, 9–13] in terms of mobility handling, it relies on the availability of the network diameter to function correctly. To resolve this issue, we introduce the second CDS protocol, namely Multi-Initiator (MI). Unlike SI, MI uses *a set of initiators* to grow a number of dominator trees and then connect the trees to form the CDS. Since MI uses SI to construct each dominator tree, it inherits the advantages of SI (e.g., mobility handling) and at the same time avoid the need of the global information. The simulation results of both static and mobile ad hoc scenarios validate that our tree-based CDS protocols construct and maintain CDS of competitive size with low overhead. An analytical model of the convergence time and the number of messages required by the SI and MI CDS protocols is presented and the results obtained from the analytical model match well with the results from the simulation.

To improve the performance of the proposed tree-based CDS protocols, we further proposed two extensions, namely Fast-Convergence Dominator Tree Construction (FC-DTC) and Extended Mobility Handling (EMH). Unlike the original tree construction protocol used in SI and MI, the FC-DTC protocol does not rely on defer timers. It not only significantly reduces the convergence time for CDS construction but also keeps all the advantages of the tree-based CDS protocols, such as small CDS, low overhead, and mobility handling mechanism. On the other hand, EMH enables SI and MI to adapt to topology changes more efficiently. It consists of two parts. The first part, namely Height-Reduction (HR), focuses on shortening the recovery time of CDS mobility handling by dynamically adjusting the height of trees; the second part, namely Initiator-Reduction (IR), keeps the competitive size of CDS by controlling the number of initiators while doing mobility handling. Simulation results show that the two extensions, FC-DTC and EMH, successfully improve the performance of SI and MI in sense of the convergence time and mobility handling.

The rest of this paper is organized as follows. The existing CDS protocols for MANETs are reviewed in Chapter 2. The two tree-based CDS protocols as well as their performance evaluation are described in detail in Chapter 3. To improve the performance of the proposed protocols, two extensions are introduced and evaluated in Chapter 4. Chapter 5 concludes this paper.

Chapter 2

Related Works

In this chapter, existing CDS protocols for MANETs are reviewed. Although Constructing the minimum CDS is known to be NP-hard [5], a number of protocols have been proposed to approximate the minimum CDS. Considering the natures of MANETs, the centralized CDS protocols [20–22] are not suitable, as they construct a dominating set under the assumption that the complete network topology is known. Therefore, in this paper, we focus on distributed CDS protocols. A number of distributed CDS protocols have been proposed for different design goals. Depending on design goals, distributed CDS protocols are categorized into the minimum CDS protocols, energy-aware CDS protocols, k-CDS protocols, and Directed CDS protocols, and each of them are presented in the following sections.

2.1 Minimum CDS Protocols

The most relevant to this paper is the minimum CDS protocols [6–11, 13], which are to construct a small CDS based on localized information. Depending on the nature of the approach, these CDS protocols can be classified into substraction-based and addition-based.

2.1.1 Substraction-based CDS Construction

The substraction-based CDS protocol begins with the set of all nodes in the network, then systematically removes nodes to obtain the CDS. The best known CDS protocols in this category include Wu's [6], Dai's [7], and Stojmenovic's [8] protocols. All of these CDS protocols consist of two phases. In the first phase, each node collects k-hop neighboring information by exchanging messages with its one-hop neighbors. If a node finds that there is a direct link between any pair of its one-hop neighbors, it removes itself from the consideration

of the CDS. In the second phase, additional heuristic rules are applied to further reduce the size of the CDS. Wu's protocol [6] uses Rule 1 and Rule 2 in its second phase, and Dai's protocol [7] generalizes this second phase by Rule k, in which a node is removed from a CDS if all its neighbors are covered by a connected set of nodes in its k-hop neighbors and its id is the smallest than any node in the set. Note that Rule k requires k-hop neighboring information, and hence the larger value of k is the more control overhead the protocol incurs. Stojmenovic's protocol [8] further reduces the size of CDS by applying two additional rules, the extended coverage condition and the key reversal techniques. The analysis and implementations presented in [23,24] show that protocols with one hop information results in a large CDS and using more than two hop incurs heavy communication overhead. Therefore, Wu's, Dai's, and Stojmenovic's protocols with two hop information are practical. Since these protocols use two-hop neighbor information, a node needs at least two beacon periods to obtain the latest topology information before it can react to any topology change caused by nodal mobility.

2.1.2 Addition-based CDS Construction

The addition-based CDS protocol starts from a subset of nodes (commonly disconnected), then includes additional nodes to form the CDS. The most famous protocols in this category are Wan [9–11], Chen's [12], and Li's [13] protocols. In general, these tree protocols obtain the CDS by expending the Maximal Independent Set (MIS). Their protocols also have two phases. In the first phase, the maximal independent set of the given network topology is constructed distributively by recursively selecting the nodes with the most neighbors locally. The nodes in the MIS become the skeleton of the CDS. Although nodes in the MIS are not connected, the distance between any pair of its complementary subsets is known to be exactly two hops away. Hence, in the second phase, a localized search is used to include additional nodes to connect the nodes in the MIS and form the CDS. The difference among [9–13] lies in how nodes in MIS are connected in the second stage (e.g., top-down or bottom-up fashion).

2.2 Energy-Aware CDS Protocols

In the energy-aware CDS protocols [14, 15], the criteria to determine the node in CDS include the energy. Such consideration is very important, as the power constraints is the primary concern in MANETs where nodes do not have power supply. Wu's energy-aware protocols [14] extends the substraction-based CDS protocols [6,7]. In [14], the energy level of a node is used as a priority to eliminate it from a CDS instead of *id*. Note that a tie can be broken by *id*. As a result, nodes with higher energy level have more likely form a dominating set. Kim's protocol [15] obtains a dominating set starting from a unique initiator. Each node in a tree set a defer timer inversely proportion to the number of uncovered nodes as well as energy level. By incorporating both the number of uncovered nodes and energy level, it prolongs the life time of a CDS with competitive size.

2.3 k-CDS Protocols

The protocols in this category are to construct a fault-tolerant k-CDS [16–18], under the assumption that the original graph is k-connected. k-CDS is defined as the CDS in which the dominating set is still connected after the removal of any $k-1$ nodes from the graph, and each node has at least k neighbors in the CDS. In other words, there exist at least k-connected paths through dominating set between any two nodes. Dai's k-CDS protocol [16] again obtains a CDS by eliminating unnecessary nodes from k-CDS. A node is removed from the dominating set if there exist k independent paths between any pair of nodes in its neighbors. *id* is used to avoid simultaneous node removal from the dominating set. Li's k-CDS protocol [17, 18] obtains by expending a MIS. First, it generate 1-connected CDS by using an addition-based protocol such as [9], and this 1-CDS is a skeleton of k-CDS. In the second phase, additional nodes are selected to make the CDS k-connected.

2.4 Directed CDS Protocols

The protocols in this category are for a special hardware setting, where each node in a netowork has different transmission range and is equipped with a directional antenna. Yang's protocol [19] creates a directional CDS in an arbitrary directed gprah, which is assumed to be strongly connected. Similar to the original substraction-based CDS protocols [6–8], Yang's protocol [19] obtains a directed CDS by eliminating unnecessary nodes and from a dominating set. In the first phase Node Coverage Condition (NCC) is used in which a node is removed from the CDS if any pair of absorbant and dominating neighbors is connected. In their work, the source node of the link is defined as an absorbant neighbor for the destination node, and the destination node of the link is defined as a dominating neighbor for the source node. In the second phase, Edge Coverage Condition (ECC) is employed where an edge is removed from the subset of edges if there exists a replacement path between the source and sink nodes of the edge. Furthermore, the authors proposed the sector optimization mechanisms to minimize the number of switch-on sectors of directional antennas by adjusting the angle of sectors, and the topology control technique to minimize the transmission power.

2.5 Metrics to Evaluate CDS Protocols

Although different types of CDS protocols have different design goals, there are several common metrics to evaluate CDS protocols. The most important metric is the size of CDS. This is because that a small size of CDS generally reduces the size of routing table or decreases the number of retransmissions. This not only alleviates the storage and communication overhead of routing protocols, but also improves the delivery rate and delay. Second, the convergence time to create a CDS is crucial to the availability of a virtual backbone. A MANET frequently experiences topology changes due to nodal mobility. CDS protocols should converge quickly as soon as possible in order to accommodate nodal mobility. Third, a dominating set is desirable to be calculated with less control overhead. A large amount of

traffic to construct and maintain a CDS causes to poor routing performance. Finally, CDS protocols must be able to adapt to topology changes due to nodal mobility or power-off. Since it takes long time to execute a whole process to reconstruct a CDS when a CDS is corrupted, protocols are required to locally recover corruption of the CDS.

Chapter 3

Tree-Based CDS Construction Protocols

3.1 The Basic Design

The idea of our proposed protocols is based on a simple greedy strategy: To obtain a smaller dominating set, a node with more neighbors should be included in the set. To reduce the communication overhead, we propose the uses of defer timers in our distributed CDS protocols. By appropriately setting the defer timer at each node, nodes with more neighbors will have higher probability to be included in the CDS. Generally speaking, our proposed tree-based CDS protocols consist of three phases: i) initiator election; ii) tree construction; and iii) tree connection, as illustrated in Algorithm 1. In the initiator election phase, a number of initiators are elected distributively. This is achieved by letting the local minimum among α-hop neighbors to be initiators. In the tree construction phase, starting from each initiator, nodes utilize the defer timer to generate disjoint dominator trees. The defer timer is set inversely proportion to the number of uncovered neighbors in order to give higher priority to nodes with more uncovered nodes. In the tree connection phase, additional nodes are identified to connect the disjoint dominator trees. Each initiator collects the information of its neighboring trees from leaf nodes in its tree, and then sends a control message to a border node to connect trees. The initiators, the dominator trees, as well as the nodes that connect the trees altogether form the CDS. Depending on how many initiators are elected, two different flavors of the tree-based CDS protocols, namely Single-Initiator (SI) and Multi-Initiator (MI), are presented in this paper.

We assume each node to have a unique *id* value, such as its MAC address. In the protocols, a node is always in one of the four possible states, i.e., *uncovered*, *covered*, *dominator*, and *dominatee*. Similar to what is defined in IEEE 802.11 wireless LAN specification [25],

Algorithm 1 Tree-Based CDS Protocol Skeleton

```
1: Initiator Election
2:    /* Initiators are localized elected */
3: Tree Construction
4:    /* Each initiator grows a dominator tree independently */
5: Tree Connection
6:    /* Additional nodes are included to connect disjoint trees */
```

each node periodically broadcasts a beacon to its neighbors every beacon period. We assume a node's beacon contains a number of fixed-length fields, including a type, the node's *id*, a *color* value, the node's current *state*, the node's initiator *id*, and the node's dominator *id*. The *announce* in the subsequent sections is a beacon with distinct type values. Before the protocol starts, each node sets its initiator *id* to $INIT_0$, its *state* to *uncovered*, and the *color* value to 0. The additional notations used in the subsequent discussion are summarized in Table 3.1.

3.2 The SI Tree-Based CDS Protocol

If the network size (i.e., network diameter) is known in advance, a unique initiator can be elected distributively in the initiator election phase within a predefined amount of time. Since only one initiator tree will be generated from the unique initiator at the end of the tree construction phase, the tree connection phase is not required. The subsequent subsections elaborate the initiator election and tree construction phases of the SI CDS protocol.

3.2.1 SI-Initiator Election Phase

In the initiator election phase, the *id* of each node is used as the metric to determine the initiator. When the $ITimer$ expires, if a node finds that its initiator *id* is $INIT_0$, it first sets its initiator to its own *id*. The reason why an initiator *id* is initialized by $INIT_0$ is to deal with asynchronous environment where nodes act autonomously (see Lemma 1). When a node receives a beacon, it compares its initiator *id* with the initiator *id* in the beacon. If a node finds that its initiator *id* is larger than the beacon's, it sets its initiator *id* to the

Table 3.1: Definition of notations.

Symbol	Definition
$id(i)$	the id of node i
$initiator(i)$	the initiator of node i
$state(i)$	the current state of node i
$dominator(i)$	the dominator of node i
$color(i)$	the current color value of node i
$ITimer$	the countdown timer for initiator election
$DTimer$	the countdown timer that determines if a node is a dominator
$CTimer$	the countdown timer that keeps track of a node's dominator
$INIT_0$	a number larger than the id of any node
$Init_{Max}$	the maximal value for $ITimer$
T_d	the defer timer used for $DTimer$ in the tree construction phase
t_d	the value used for $CTimer$ to keep tracks of the presence of a node's dominator
T_{max}	the maximal value for $DTimer$
n_{uc}	the number of uncovered neighbors
β	the coefficient used in the defer timer
δ	the threshold for color value between neighbors
$H(i,j)$	the number of hops between node i and j
n	the number of nodes in the network
n_{init}	the average number of initiators in the network
S	the size of the field the network is deployed
$Diam$	the diameter of the network
r	the transmission range of a node
Δ	the average number of neighbors
$\overline{d}$	the average distance between two neighbors
α	the range in number of hops to elect a local minimum

initiator *id* in the beacon. A node can switch its *state*, if it does not receive beacon from nodes with smaller *id* than it during the period of twice of $Init_{Max}$. This duration is long enough so that at the end of the election phase the smallest *id* among all nodes will be propagated to all nodes in the network. Note that if a new node with the smaller *id* than any nodes in the network joins during the initiator election phase, it takes trice of $Init_{Max}$ (detail is shown in Lemma 1).

Algorithm 2 describes the SI initiator election phase. Note that in the pseudo code, the initiator will send out *announce* beacon in every beacon period. The primary job of *announce* beacon includes detecting initiator failure and a disruption of a tree. For detecting initiator failure, *announce* beacons refresh the $ITimer$ of other nodes which expires after twice of the $Init_{Max}$ beacon periods. Because *announce* beacons are originated from the initiator, if the initiator leaves the network or there is a partition in the network, the $ITimer$ of some node will expire. In this case, the node will wait twice of the $Init_{Max}$ beacon periods to make sure the $ITimer$ of every node in the network expires and restart the initiator election process. For detecting a disruption of a tree, each initiator keeps increasing *color* value by one at every beacon period and each node in its tree updates color value accordingly. The disconnection caused by nodal movement interrupts the propagation of *announce* message, which will eventually lead to large color differences. When a node receives a beacon signal from its dominator indicating a large color difference, it concludes that there is a disconnection with its dominator.

3.2.2 SI-Tree Construction Phase

When $ITimer$ expires, a node moves to the tree construction phase. When a node finds that it is the initiator, it immediately switches its state to *dominator* and its initiator to itself. When an uncovered node receives a beacon from a dominator neighbor for the first time, it sets its dominator to that neighbor, its initiator to that neighbor's initiator, switches

Algorithm 2 The SI initiator election

```
1: /* node i in the network executes the following: */
2: INITIALIZATION:
3:    initiator(i) ← INIT_0
4:    state(i) ← uncovered
5:    color(i) ← 0
6:    DTimer(i) ← −1
7:    send out announce
8:    ITimer(i) ← Init_Max, start ITimer(i)
9:
10: WHEN NODE i's ITimer(i) EXPIRES:
11: if initiator(i) = INIT_0 then
12:    /* initial announcement */
13:    initiator(i) ← i
14:    send out announce
15:    ITimer(i) ← 2 × Init_Max, start ITimer(i)
16: else if initiator(i) = i then
17:    /* node i is an initiator */
18:    color(i) ← color(i) + 1
19:    send out announce
20:    ITimer(i) ← Init_Max, start ITimer(i)
21: else if initiator(i) ≠ i then
22:    /* re-elect initiator */
23:    state(i) ← uncovered
24:    initiator(i) ← INIT_0
25:    color(i) ← 0
26:    ITimer(i) ← 2 × Init_Max, start ITimer(i)
27: end if
28:
29: ON RECEIVING announce FROM NODE j:
30: if initiator(i) > initiator(j) then
31:    initiator(i) ← initiator(j)
32: else if initiator(i) = initiator(j) && color(i) ≠ color(j) then
33:    color(i) ← color(j)
34:    send out announce
35:    ITimer(i) ← 2 × Init_Max, start ITimer(i)
36: end if
```

its *state* to *covered*, and sets a defer timer T_d in inversely proportion to the number of its uncovered neighbors using Equation 3.1.

$$T_d = \begin{cases} \frac{T_{max}}{(n_{uc})^{\beta}} & if\ n_{uc} \geq 1 \\ T_{max} & if\ n_{uc} < 1 \end{cases} \tag{3.1}$$

As long as $\beta > 1$, Equation 3.1 allows nodes with more uncovered neighbors to be given a smaller timer value, hence their defer timer will expire earlier. When a node's defer timer expires, if the node still has uncovered neighbors, it switches its *state* to *dominator* to

cover them. Otherwise, it switches its *state* to *dominatee*. The number of uncovered nodes, denoted as n_{uc}, can be learned from the beacon of one-hop neighbors. Note that the scale of T_d is much larger than the beacon periods. Therefore, even if each node sends out its beacon asynchronously, our differ timer scheme guarantees that a node with more uncovered neighbors has shorter timer in most of cases.

At the end of the tree construction phase, all nodes will be in either *dominator* or *dominatee* state. The set of dominators forms a tree, which is referred to as the dominator tree. If there is only one initiator and the network is connected, the dominator tree will be the CDS for the entire network. We formalize the SI tree construction phase in Algorithm 3.

In Algorithm 3, if a covered node i does not receive a beacon from i's dominator for t_d beacon periods, it implies that i's dominator has left and node i can then switch its *state* to *uncovered*. The assignment of t_d depends on the message error rate. For the 802.11 network, a small value such as 4 is sufficient [26]. $CTiemer$ is used to track the dominator. A node updates $CTimer$ if it receives a beacon from its dominator and the dominator is still in *dominator* state. If $CTimer$ expires, it implies that its dominator leaves from the network.

3.2.3 Correctness of the SI Protocol

In this subsection, we prove that the SI CDS protocol generates a CDS for a given network topology as long as the topology remains connected and stable for a period of time (the time required to construct a CDS) and the value of $Init_{Max}$ is larger than the network diameter.

Lemma 1 *A unique initiator is elected within* $3Init_{Max}$ *in the asynchronous environment where nodes act autonomously, as long as* $Init_{Max}$ *is larger than the network diameter, Diam, and the topology is stable for a period of time.*

Proof 1 *Assume the network consists of several nodes. Among them, node i has the smallest id. At time* t_1*, some nodes in the network start the initiator election phase. Let node j joins the network at time* t_2 *where* $t_1 < t_2$*. In the case of* $id(i) \leq id(j)$*, each of i's neighbors*

Algorithm 3 The SI tree construction

/* initiator i executes the following */
NODE i **IS ELECTED AS THE INITIATOR:**
  $state(i) \leftarrow dominator$
  $color(i) \leftarrow color(i) + 1$ each time before node i sends out beacon

/* node i executes the following when it receives a beacon */
ON RECEIVING A BEACON FROM NODE j**:**
if $state(j) = dominator$ **then**
  if $color(i) < color(j)$ **then**
    $color(i) \leftarrow color(j)$
  end if
  if $state(i) = uncovered$ **then**
    $state(i) \leftarrow covered$, $dominator(i) = j$
    $CTimer(i) \leftarrow t_d$, start $CTimer(i)$
  end if
  if $state(i) = covered$ || $state(i) = dominatee$ **then**
    if $dominator(i) = j$ **then**
      $CTimer(i) \leftarrow t_d$, start $CTimer(i)$
    end if
    if $\exists$ i's neighbor k, $|color(j) - color(k)| > \delta$ **then**
      $state(i) \leftarrow dominator$. $dominator(i) \leftarrow j$
    end if
  end if
end if

/* node i executes before it sends out a beacon */
WHEN NODE i **IS IN** *covered* **OR** *dominatee* **STATE:**
if node i has no *uncovered* neighbor **then**
  $state(i) \leftarrow dominatee$
else if node i has at least one uncovered neighbor **then**
  compute T_d
  if $DTimer(i) > T_d$ **then**
    $DTimer(i) \leftarrow T_d$, start $DTimer(i)$
  end if
end if

WHEN NODE i **IS IN** *dominator* **STATE:**
if (node i has no *covered* neighbor) && (node i has at least one *dominator* neighbor) **then**
  $state(i) \leftarrow dominatee$, $dominator(i) \leftarrow$ one of node i's dominator
  $DTimer(i) \leftarrow -1$, stop $Dtimer(i)$
end if

/* node i executes when a countdown timer expires */
WHEN $DTimer$ **EXPIRES**
if node i has *uncovered* neighbors **then**
  $state(i) \leftarrow dominator$
end if

WHEN $CTimer$ **EXPIRES**
  $state(i) \leftarrow uncovered$

will set its initiator to i after receiving a beacon from i. Their initiator value will stay as i because $id(i)$ is the smallest among all nodes in the network. In the next round of beacon exchanges, i's neighbors will propagate their initiator information further to reach nodes two hops away from i, which again will allow more nodes to set i as their initiator. This process is repeated until all nodes in the network set their initiator as i. At the end, node i is elected as the initiator as long as $id(i) < id(j)$ for any additional new node j in $Init_{Max} + \max_{\forall k} H(i,k) \leq Init_{Max} + Diam \leq 2Init_{Max}$. Thus, the above claim is true. In the case of $id(i) > id(j)$, depending on the value of t_2, we further break it into the following two subcases.

If $t_2 \geq t_1 + H(i,j)$, when node j receives announce from node i at the time $t_1 + Init_{Max} + H(i,j)$, initiator$(j)$ is still $INIT_0$ because $ITimer(j)$ has not yet expired. According to the protocol, $id(i) < INIT_0$ so node j set its initiator id to be i's. In other words, all nodes will set their initiator id to i's within $Init_{Max} + \max_{\forall k} H(i,k) < 2Init_{Max}$. Thus, the above claim is true.

If $t_2 < t_1 + H(i,j)$, the $ITimer$ of node j will expire before it receives announce from node i. In this case, node j will send out its own announce message to its neighbors. The nodes that received node i's announce in the past override their initiator id to be j's as $id(i) > id(j)$. Node j will be elected as the initiator in $t_2 - t_1 + Init_{max} + \max_{\forall k} H(i,k) < 3Init_{Max}$. Therefore, the above claim is true.

Lemma 2 *A new initiator will be selected by the SI initiator election phase within bounded beacon periods if the original initiator leaves the network assuming the network remains connected and stable.*

Proof 2 *From the pseudo code of the SI initiator election phase, we can see that the initiator will send out announce messages every $Init_{Max}$ beacon periods. Other nodes only forward announce messages if its initiator id is bigger than the initiator's in the message or the color*

value is different. If the initiator is active, its color value will change every $Init_{Max}$ beacon periods, the refresh announce will keep the ITimer of other nodes from expiring.

When the original initiator leaves the network, the ITimer of all nodes in the network will expire since no more refresh announce message is coming. After ITimer expires, each node sets all the information back to the initial value and the initiator election phase will start over again. According to Lemma 1, a unique initiator will once again be selected.

In the following theorem, we prove that our SI protocol successfully obtains the connected dominating set for the given network topology.

Theorem 3 *If the network topology remains connected and stable for a period of time, the collection of all dominator nodes resulted from the SI protocol forms a connected dominating set for the entire network.*

Proof 3 *We claim that if the network topology is connected and unchanged, eventually all the dominator nodes will form a dominating set and the induced graph is connected.*

We prove by induction on the number of nodes n.

Induction base: when $n = 1$, the only node is the initiator. The initiator enters dominator after the initiator election phase, which is a connected dominating set. The above claim is true.

Induction hypothesis: Assume when $n = k - 1$, the CDS generated by the SI protocol covers all $k - 1$ nodes. Let us denote the graph with $k - 1$ nodes as $G(k - 1)$.

Induction step: Now one additional node (node k) joins the network. Let us denote the resulting network as $G(k)$. If one of node k's neighbors is a dominator, the CDS covering $G(k - 1)$ will cover $G(k)$ and is still a CDS.

If none of k's neighbors is a dominator, all neighbors of node k must be covered by some dominators in the CDS of $G(k - 1)$. According to the SI protocol, all node k's neighbors will go through the covered state and set up the defer timer appropriately. When the first one of them has its defer timer expires, that neighbor will enters the dominator state to cover node

k. After that, there will be no more uncovered node and therefore we have a dominating set. Since the new dominator node enters the dominator state from the covered state, it must have a dominator neighbor. (A node enters covered state only after it receives a beacon from a dominator neighbor.) In other words, the nodes in the dominating set, after including the additional dominator node, are still connected. The CDS covering $G(k-1)$ plus the new dominator node will be the CDS covering $G(k)$.

If a node can not correctly obtain n_{uc} due to the collisions, the size of CDS increases as nodes with fewer uncovered nodes set shorter timer. However, this never affects the correctness of the SI protocol. Thus, the SI protocol successfully obtains a CDS, as long as each node has bidirectional links between its neighbors.

3.2.4 SI Mobility Handling

Now we would like to show that the SI protocol can adapt to nodal mobility. This is done by showing that the SI protocol successfully maintains the CDS under changes of the network topology, which are composed of the following four cases.

1. The initiator leaves the network.
2. A new node joins the network after the construction of the CDS.
3. A redundant dominator node switches to the *dominatee* state while still keeping the dominating set connected.
4. A dominator node leaves the network.

Case 1 is proved in Lemma 2, and Case 2 is exactly the situation in Theorem 3. From the SI tree construction pseudo code, we can see that if a dominator does not cover any neighbor and has at least one *dominator* neighbor, it will switch its state to *dominatee* and set its dominator field as the *id* of the dominator neighbor. By doing so, the total number

of *dominator* nodes is reduced and we have maintained the CDS under Case 3. This could happen after a series of node movements.

Case 4 is handled by the color scheme. From the SI tree construction pseudo code, we can see that a node changes its color only after receiving a larger color value from a dominator neighbor. Since the initiator keeps increasing its color value, if the CDS is intact, the color difference of neighbor nodes should be very small. If the CDS is broken into disconnected segments, the segment that contains the initiator will keep increasing its color value while other segments will have the color value unchanged. When a node sees two neighbors with large color difference, it concludes that it is a border node between the disconnected segments and enters the *dominator* state. This process will continue until a CDS is reconstructed. The threshold value of color difference (i.e., δ) is decided by the moving speed of nodes and the diameter of the network. It should be large enough to distinguish the node movement from network disconnections.

3.2.5 Example of the SI Protocol

Figure 3.1 (a) shows the snapshot of a network topology. In Figure 3.1 (a), a dotted circle represents a node in *uncovered* state and a dotted line represents a link between two nodes. Figure 3.1 (b) shows the result of the initiator election phase. According to the SI protocol, node 1 is elected as the initiator, which is denoted by a shaded square. The neighbors of node 1 that are switched to the *covered* state are denoted by a double dotted circle. In the next beacon period, node 3 finds that it has no uncovered neighbor and then switches to the *dominatee* state, which is denoted by a circle in Figure 3.1 (c). At the same time, node 2 and 4 set their $DTimer$ and start counting down the timer. Since node 2 has more uncovered neighbors, it has a shorter timer and its timer will expire before node 4's. As shown in Figure 3.1 (d), node 2 will switch to the *dominator* state, which is denoted by a gray circle. After that, node 5 and 6 do not have uncovered neighbors and will then switch to the *dominatee* state. On the other hand, node 7 will start its own $DTimer$, as shown

in Figure 3.1 (e). While node 4 and 7 have the same number of uncovered neighbors, the timer of node 4 expires sooner as it started its *DTimer* earlier. When the timer of node 4 expires, node 4 will switch to the *doinator* state to cover node 8, as shown in Figure 3.1 (f). Afterwards, node 7 and 8 find that they do not have uncovered neighbors and switch to the *dominatee* state, which is shown in Figure 3.1 (g). At the end, the collection of the initiator and dominators form a CDS for the whole network.

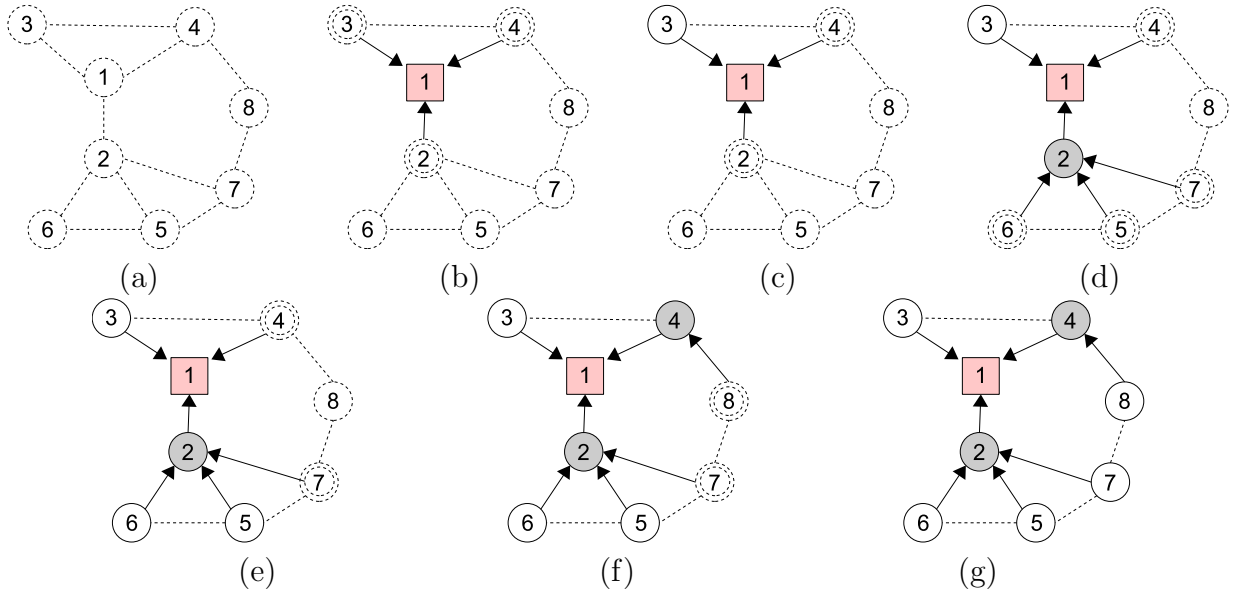

Figure 3.1: An example of the SI protocol.

3.3 The MI Tree-Based CDS Protocol

As discussed in Section 3.2, the SI protocol is able to maintain CDS in the presence of topology changes. However, SI requires the knowledge of the diameter of the network to properly set up the $Init_{Max}$. Although the network diameter may be estimated by the size of the region the network is deployed and the transmission range of a node, the estimation is not accurate. In addition, while SI is able to recover the CDS locally for most CDS breakdowns, the whole CDS will still have to be reconstructed from scratch should the unique initiator fail. A natural approach to resolve these issues is to elect multiple initiators. Each initiator generates a tree in exactly the same manner the SI protocol does. The tree construction

phase completes when all nodes are covered by dominators. This will produce several small disjoint trees and the union of these trees will form a dominating set. To obtain a CDS, we can simply include a few more nodes to connect these dominator trees. This is the basic idea of our Multi-Initiator CDS protocol (MI). If the CDS is constructed this way, the failure of an initiator will only affects the corresponding dominator tree. The other part of the CDS will remain intact. In the following sections, we explain how the initiator election phase and the tree connection phase of MI are accomplished effectively and distributively in detail. Note that the tree construction phase of MI is omitted because it is basically the same as that of SI described in Section 3.2.2.

3.3.1 MI-Initiator Election Phase

The most intuitive idea to elect multiple initiators effectively and distributively is to let the local minimum be the initiators. By listening to beacons, a node obtains the *id*s of its one-hop neighbors without introducing additional messages. If a node finds that its *id* is the smallest among its one-hop neighbors, it sets itself as an initiator.

The problem with this approach is that it may produce too many initiators. Given the average number of neighbors to be Δ, each node has the opportunity to be an initiator with the probability $\frac{1}{\Delta+1}$. Let n be the number of nodes uniformly distributed in a region of size S, and r be the transmission radius of each node, the average number of initiators n_{init} can be obtained by Equation 3.2.

$$n_{init} = n \cdot (\frac{1}{\Delta + 1}) \approx \frac{S}{\pi r^2} \tag{3.2}$$

For instance, a network of 150 nodes with $150m$ transmission radius deployed in $1000m \times 1000m$ area will have an average of 14 initiators. Too many initiators will consequently lead to a large CDS due to two reasons. First, all initiators will eventually be included as part of the CDS. Second, after each initiator generates a dominator tree, additional nodes will

be added to connect these trees. More initiators means more trees, and thus requires more nodes to connect them.

To reduce the number of initiators, the local minimum among α-hop neighbors can be elected as the initiators. As an example, to elect the local minimum among two-hop neighbors, each node can further encode the minimal *id* among its one-hop neighbors in its beacon. A node becomes an initiator only when its *id* equals to the minimal *id* of all its one-hop neighbors. With few rounds of beacon exchanges, the local minimum among two-hop neighbors will become the initiator. In this approach, no extra message is needed and the time to elect initiators is much shorter than the time required by the SI protocol in the initiator election phase. Using the same notations, the average distance between two neighbors, $\overline{d}$, is obtained by Equation 3.3.

$$\overline{d} = \int_0^r \frac{2\pi x \cdot x}{\pi r^2} dx = \frac{2}{3} r \tag{3.3}$$

The average number of two-hop neighbors is $\frac{n\pi(r+\overline{d})^2}{S}$. Therefore, the average number of initiators n_{init} can be obtained by 3.4.

$$n_{init} = n \cdot \left(\frac{1}{\frac{n\pi(r+\overline{d})^2}{S} + 1}\right) \approx \frac{9}{25} \cdot \frac{S}{\pi r^2} \tag{3.4}$$

For instance, the network of 150 nodes with $150m$ transmission radius deployed in $1000m \times 1000m$ area now has an average number of 5 initiators. This significantly reduces the possibility of a large CDS.

The multi-hop local minimum idea can be extended to α-hop. Since the average number of α-hop neighbors is $\frac{n\pi(r+(\alpha-1)\cdot\overline{d})^2}{S}$, the expected number of local minimum can be estimated by Equation 3.5. Note that when α is equal or larger than the network diameter, only one initiator will be elected in the network and the protocol is reduced to SI.

$$n_{init} = max\{n \cdot \left(\frac{1}{\frac{n\pi(r+(\alpha-1)\cdot\overline{d})^2}{S} + 1}\right), 1\} \tag{3.5}$$

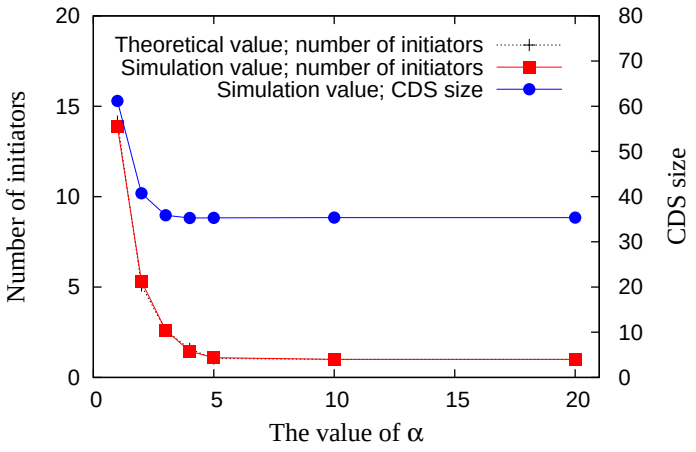

Figure 3.2: The number of initiators and CDS size.

Figure 3.2 shows the number of initiators and the CDS size with respect to the value of α in the network of 150 nodes with $150m$ transmission radius deployed in $1000m \times 1000m$ area. As illustrated in Figure 3.2, the number of initiators as well as the size of CDS decrease as the value of α increases. In the case $\alpha = 2$, the size of CDS is competitive and each dominator tree is small and easy to maintain. Therefore, in the rest of the paper, the α value of the MI protocol is set to 2.

3.3.2 MI-Tree Connection Phase

In this phase, additional nodes are included to connect neighboring dominator trees. If a node has a neighbor belonging to a different initiator, it is referred to as a border node. To connect the dominator trees distributively, the most intuitive idea is to have all the border nodes turn into dominators. Although this approach does not introduce extra messages, it will create a very large CDS as there are possibly many border nodes between each pair of neighboring dominator trees. To limit the size of CDS, it is better that the root of a tree (i.e., initiator) determines what border nodes are utilized to connect the neighboring trees.

Since an initiator does not know what neighboring trees it has with only the localized information, extra messages have to be introduced so that the initiator can collect such information from its border nodes. After the defer timer expires, if a node finds that it is a border node, it sends a message to its initiator which includes the initiator's *id* of the

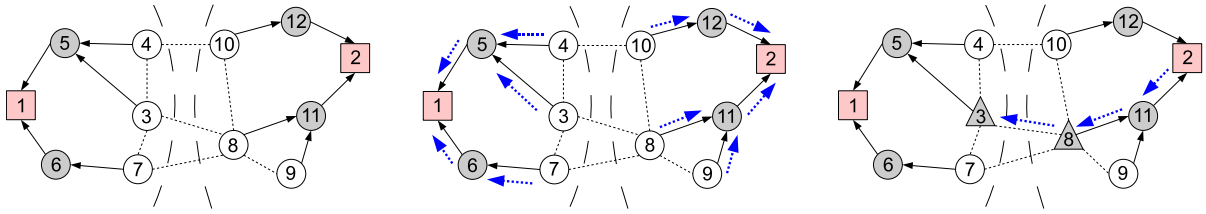

Figure 3.3: An example of the MI protocol.

neighboring tree. Figure 3.3 (a) depicts a snapshot of the network after the tree construction phase completes. For example, border node 3 belonging to the tree rooted at the initiator 1 finds that its neighbor 8 belongs to the tree rooted at the initiator 2 from node 8's beacon. Node 3 then sends a message to node 5 containing the *id* of the initiator 2. Similarly, node 4, 7, 10, 8 sends a message to their initiator containing the initiator *id* of their neighboring tree.

At the first glance, this process seems to introduce many messages. However, when a dominator receives messages from the neighbors it covers about the neighboring trees, it only forwards one copy of the messages if the initiator *id* contained in the messages are the same. For instance, in Figure 3.3 (b), when node 5 receives messages from node 3 and 4 about the same neighboring dominator tree, it only forwards one copy of the message to its dominator node 1. By doing so, the number of messages can be controlled to $O(n \cdot m)$ where n is the number of nodes in the network and m is the number of neighboring dominator trees.

When an initiator learns about its neighboring trees, it can then instruct only border nodes on a particular path to each of its neighboring trees to switch its *state* to *dominator* and connect to its neighboring trees. The border nodes that are used to connect the trees are referred to as the bridge nodes. Any metric can be used to elect bridge nodes among border nodes. In this paper, the node with the smallest *id* is elected as bridge nodes. For example, in Figure 3.3 (c), initiator 2 elects node 8 as a bridge because it has the smallest *id* among border nodes which connects the neighboring tree 1. Our CDS consists of the dominator nodes in the dominator trees and the bridge nodes that connect the trees.

If each initiator tries to connect to each of its neighboring dominator trees, it is likely that there will be two paths between each pair of neighboring dominating trees. In the worst case, at most four border nodes (two for each path) will become dominators. While having two paths between neighboring dominator trees may improve the degree of fault tolerance and system throughput, it will create a larger CDS. To limit the size of a CDS, an initiator makes a connection to a neighboring dominator tree only when the *id* of the initiator of the neighboring tree is smaller than its own. This roughly reduces the number of the bridge nodes by half.

3.3.3 Correctness of the MI Protocol

Since MI is built on top of SI, most of the theorems in Section 3.2.3 directly apply to MI. The only additional issue is whether or not the tree connection phase can successfully connect the dominator trees.

Theorem 4 *If the network topology remains connected and stable for a period of time, the collection of all dominator nodes resulted from the MI protocol forms a connected dominating set for the entire network.*

Proof 4 *After the initiator election and tree construction phases, a set of the disjoint dominator trees will be created. The nodes in these dominator trees together form a dominating set. The neighboring dominator trees will be at most three hops away. It is because if two neighboring dominator trees are more than three hops away, one of the nodes will not be covered by any dominator. For example, in Figure 3.4, the neighboring trees are distanced by four hops. This will leave node 5 to be uncovered. However, this situation should not happen because according to the tree construction protocol, when the DTimer of node 4 and 6 expire, they will find node 5 as an uncovered neighbor and therefore one of them will switch to dominator to cover node 5. In other words, there will be at most two adjacent covered nodes and each of them has a different initiator. These two border nodes will report to their*

initiators and, according to MI, the initiator with larger id will then initiate the connection to the neighboring tree, providing that the network topology remains connected and stable for a period of time.

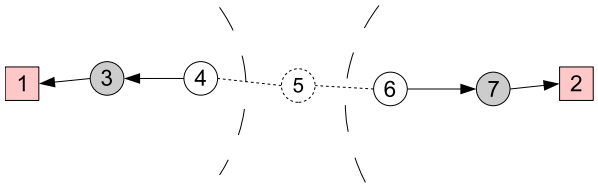

Figure 3.4: An impossible case at the end of MI tree construction phase.

3.3.4 MI Mobility Handling

In MI, each dominator tree is maintained in the same manner as SI. This allows the MI protocol to take advantage of the mobility handling capability of SI inside each of the dominator trees. The root of a dominator tree periodically broadcasts the *announce* message so that any topology changes due to mobility can be captured and handled in a timely fashion. Therefore, MI can naturally handle the four different mobility cases mentioned in Section 3.2.4. However, since MI will create multiple dominator trees, there are additional mobility cases that involve more than one tree. Notice that most of these new cases can be considered as the combinations of the aforementioned four cases. For instance, if a dominator moves from one tree to another, it can be seen as Case 4 for the first tree plus Case 2 for the second tree. Consequently, most of these new cases can be handled and resolved properly and locally with no change to the protocol. The only new case that needs to be addressed is when a bridge node leaves its dominator tree.

Given a bridge node x, assume that it belongs to a dominator tree with root a, and x is used by a to connect to a neighboring dominator tree with the root b. Clearly, both a and b are initiators. If x leaves the tree, the dominator that covers x, say node p, will find out after a couple of beacon periods. In this case, p will first try to fix the problem locally

by querying its neighbors if any of them has a neighbor belonging to the initiator b. If p receives a positive response from some of its border neighbors, it instructs one of them to switch to dominator state i.e., act as the new bridge between two trees. If p does not hear back from its neighbors for a period of time, it sends a message to the initiator a, which will send a tree-wide query down to all its border nodes looking for a possible connection to the neighboring dominator tree rooted at b. The responses sent back from the border nodes are handled similarly to the messages in the tree connection phase. Afterwards, if a receives some responses from the border nodes with neighbors belonging to initiator b, it instructs the border nodes on one of the paths to turn to dominators (i.e., become bridge nodes) to connect two trees. If a does not receive any response, that implies that the dominator tree rooted at b is no longer a neighboring dominator tree for a. In this case, nothing needs to be done by node a.

It may seem odd that the dominating set will remain connected if nothing is done after the departure of a bridge node makes two neighboring dominator trees no longer neighbors to each other. If we regard each dominator tree as a big cluster, what the tree connection phase is doing is to create an edge between each neighboring clusters. If two clusters (trees) are no longer neighbors to each other, as long as the whole network remains connected, they should be connected via the other clusters (trees). If the departure of a bridge node partitions the network into components, it will be impossible to create a CDS for the network. However, the MI CDS protocol can still maintain a CDS inside each of the connected components and recover back to a single CDS when the components reconnect to each other. This feature is especially important for MANETs.

3.4 Simulations

To evaluate the performance of SI and MI, we implement our protocols in C++ and ns-2 [27] along with the other CDS protocols, including Dai's with two-hop neighboring

Table 3.2: Simulation parameters.

Parameters	Value
Simulation area	$1000m \times 1000m$
Number of nodes	100 to 450
Transmission range	$150m$
1 beacon period	1 second
T_{max}	100 beacon periods
β	1
$Init_{Max}$	20 beacon periods
α	2
Mobility model	WWP
Percentage of mobile nodes	20% to 80%
Node speed	up to $5m/s$
Number of simulations	1000

Table 3.3: Parameters for ns-2.

Parameters in ns-2	Value
Propagation	Two-Ray Ground
MAC protocol	IEEE 802.11
Data rate	2 Mbps
Traffic	CBR
Number of flows	5 concurrent flows
Number of packet	5 packets/flow
Inter-arrival time	0.25 second
Packet size	128-byte
Percentage of mobile nodes	50% or 100%
Number of simulations	100

information [7], Stojemenovic's with two-hop neighboring information [8], and Wan's [11]. In this section, the simulation results of different CDS protocols are reported and analyzed.

3.4.1 Simulation Configurations

The network topology is randomly generated by placing nodes in a $1000m$ by $1000m$ square field according to uniform distribution. If the generated network is partitioned, it is discarded and a new network topology is generated to ensure the connectivity of the whole network. The transmission range of a node is set to be $150m$. Two different network scenarios, static networks and mobile networks, are considered and simulated to evaluate the performance of CDS. For a given simulation configuration, 1000 different network topologies are generated.

Static Networks - In this scenario, the average node density changes as we change the total number of nodes. The total number of nodes placed in the field ranges from 100 to 450, which corresponds to the node density ranging from approximately 5 to 30 neighbors per node. For SI and MI, the T_{max} and β in the tree construction phase are set to be 100 beacon periods and 1. According to [26], the value of $Init_{Max}$ for SI is set to be 20 to accommodate beacon collisions. On the other hand, α for MI is set to be 2, i.e., two-hop local minimum will be elected as initiators.

Mobile Networks - In this scenario, the average node density is set to be 10 neighbors per node, and some nodes are assumed to be mobile. The percentage of the mobile nodes ranges from 20% to 80% with speed up to $5m/s$. The Weighed Way Point (WWP) [28] is adopted as our mobility model. In WWP, the weight of selecting next destination and pause time for a node depends on both current location and time. The value of weights is based on empirical data carried out on the University of Southern California's campus [29]. For Dai's, Stojmenovic's, SI and MI, the corrupted CDS is recovered according to their mobility handling procedures when the topology changes. Each simulation lasts 1000 rounds of beacon periods. In the simulation, if the network topology is partitioned into disjoint connected components, CDS protocols maintain separate CDS within each component. In the limited mobility environment, 50% nodes are mobile with $5m/s$, and in the high mobility environment, all nodes are mobile. In AODV with MI, only nodes in a CDS forward a route request (RREQ) packet. The propagation model used in ns-2 routing simulations is the two-ray ground model. IEEE 802.11 is used as the MAC layer protocol, and the data rate is 2 Mbps. For each network realization, 5 pairs of source and destination are randomly selected. A new pair of source and destination will be selected if they are not connected. Each source node generates constant bit-rate (CBR) traffic flows to its destination simultaneously. Each CBR flow sends 5 consecutive packets of 128 bytes. The inter-arrival time of packets is set to be 0.25 second.

To assess the performance of different CDS protocols, five metrics are used, including the size of CDS, the number of extra messages, the average traffic, the convergence time, and the percentage of time CDS is alive. To assess the performance of routing with CDS protocols, three metrics are used, including delivery rate, end-to-end delay, and the number of RREQ packets. For MI, the messages in the tree connection phase and query/response messages in the mobility handling are counted as extra messages. For SI, all the information exchanged between nodes are done by beacons. For Dai's and Stojmenovic's protocols, beacons are considered as extra messages since the size of the beacon increases in proportion

to the node density and is too large when compared with the standard beacon frame. Each protocol changes the beacon frame format to include additional information. For SI, node *id*, *state*, *color*, and dominator *id* are included in the beacon. MI enlarges the beacon of SI to include the initiator *id* and the minimal *id* of one-hop neighbors. The beacon of Dai's and Stojmenovic's protocols include node *id*, *state*, *marker*, and the list of *id*s for one-hop neighbors. For Wan's protocol, dominator *id* and *color* are added to the beacon. The extra bit in the beacon and the size of extra messages for each protocol are counted toward the traffic (in bps) for CDS construction. The period of time for CDS protocols to complete is defined as the convergence time in the number of beacon intervals. For SI, the initiator election is assumed to be completed in 20 rounds of beacon intervals. For MI, the initiator election is done in two rounds of beacon intervals. For the mobile network scenario, the total amount of time CDS is valid divided by the total simulation period is defined as the percentage of time CDS is alive.

3.4.2 Simulation Results for Static Network Scenario

In this subsection, the simulation results of different CDS protocols in the static network scenario are presented.

Figure 3.5 shows CDS size of different protocols with respect to the node density. It is clear that SI consistently generates the smallest CDS and Dai's consistently generates the largest CDS among all the protocols. Stojemenovic's protocol creates a smaller CDS than that of Dai's, but compared with SI and MI it generates a larger CDS. While it is proven that the size of Wan's CDS is sub-optimal [9], the size of the CDS of MI is smaller than Wan's and is very close (within few nodes) to SI. In addition, the CDS size in SI and MI remains constant when the node density increases. This suggests that SI and MI are scalable.

Figure 3.6 demonstrates the number of extra messages with respect to the node density. As illustrated in Figure 3.6, MI introduces only 30% of the number of extra messages introduced by Dai's and Stojemenovic's. Compared with Wan's, MI reduces the number of

extra messages up to 60% when the node density ranges from 5 to 15 neighbors per node. As discussed in Section 3.2, SI does not introduce any extra message.

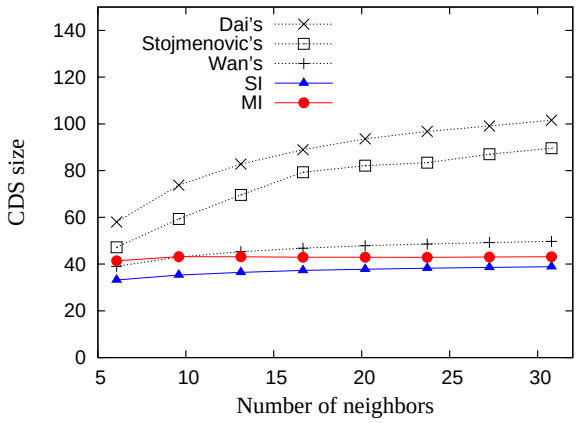

Figure 3.5: CDS size.

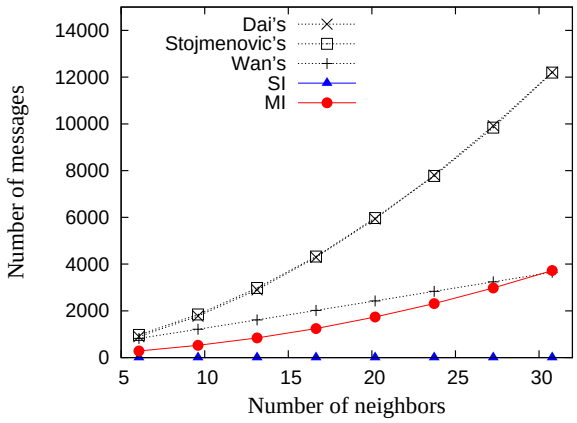

Figure 3.6: Number of messages.

Figure 3.7 shows the average traffic required for CDS construction with respect to the node density. Obviously, Dai's and Stojmenovic's protocols produce the largest traffic, and the traffic increases in proportion to the node density. This is because of the inclusion of the neighboring list in the beacon frame. For the other three protocols, the average traffic is almost the same. It is interesting that for Wan's and the MI protocols, even the number of messages increases in proportion to the number of neighbors as shown in Figure 3.6, the traffic remains relatively constant to the node density. This implies that the traffic is mostly dominated by the extra bits in the beacon.

Figure 3.8 presents the convergence time with respect to the node density. As shown in Figure 3.8, the convergence time of SI and MI decreases quickly as the node density increases. This indicates that the convergence time of SI and MI is dominated by the time of the tree construction, in which a node in dense networks is likely to have more uncovered neighbors and will have a smaller defer timer. MI takes longer time to form CDS than SI. This is due to the additional time for the tree connection phase.

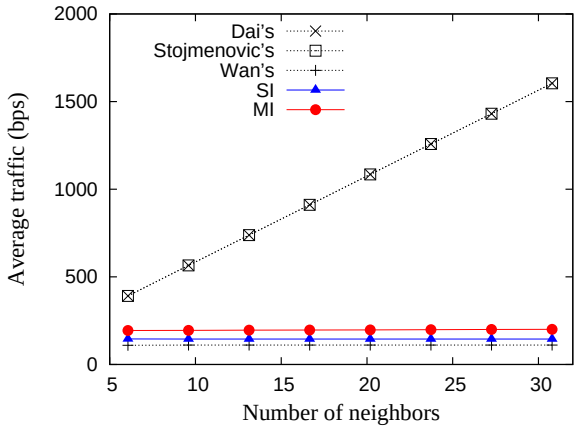

Figure 3.7: Average traffic (kbps).

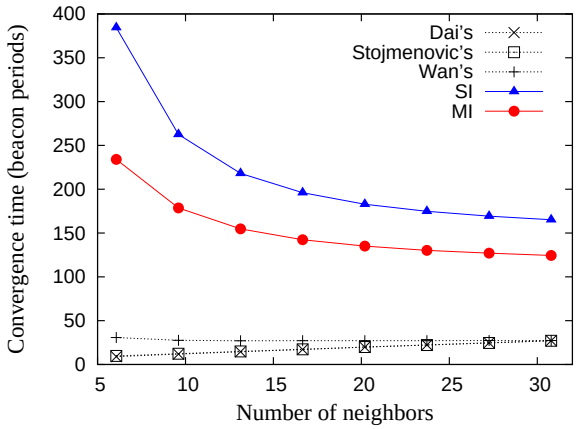

Figure 3.8: Convergence time.

3.4.3 Simulation Results for Mobile Network Scenario

In this subsection, the simulation results of different CDS protocols in the mobile network scenario are presented.

Figure 3.9 illustrates the percentage of time that CDS is alive with respect to the percentage of mobile nodes. As can be seen in Figure 3.9, MI has the highest percentage of CDS alive time than other CDS protocols except when the percentage of mobile nodes is more than 60%. Although smaller CDS is generally more vulnerable to topology changes, MI shows excellent mobility adaptation compared with the other protocols. Only when the percentage of mobile nodes is greater than 65% the percentage of time CDS is alive of MI becomes slightly lower than that of Dai's due to MI's smaller CDS size. Stojmenovic's protocol maintains a CDS for fewer percentage of time than MI and Dai's. As pointed out in [9], the time complexity of mobility recovery at each node of Dai's is as high as $O(\Delta^2)$. While Stojmenovic's protocol requires $O(\Delta)$ to maintain a CDS, it is vulnerable to nodal mobility as it further reduces the size of CDS than Dai's protocol. Thus, Dai's and Stojmenovic's protocols take more time to recover than MI. SI has smaller percentage of CDS alive time than MI, Dai's, and Stojmenovic's. This is because of the possible failure of the unique initiator, which results in the reconstruction of the whole CDS from scratch.

Figure 3.10 shows the average CDS size with respect to the percentage of the mobile nodes. As illustrated in Figure 3.10, SI consistently produces the smallest CDS and Dai's protocol consistently produces the largest CDS. The size of CDS by Stojmenovic's protocol is larger than MI except when the percentage of mobile nodes is more than 60%. The average CDS size of MI is 5% to 35% higher than Wan's. However, as can be seen in Figure 3.9, Wan's protocol is vulnerable to the nodal mobility.

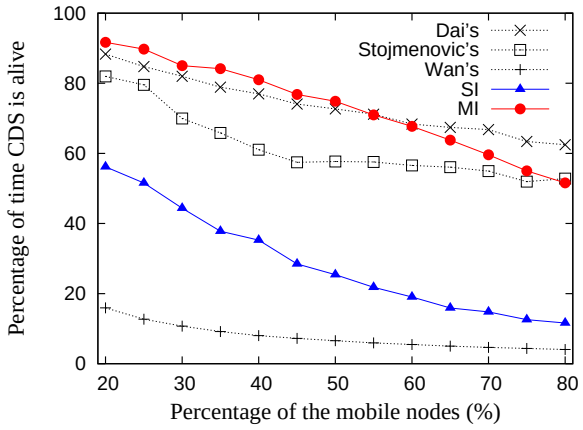

Figure 3.9: Percentage of time CDS is alive.

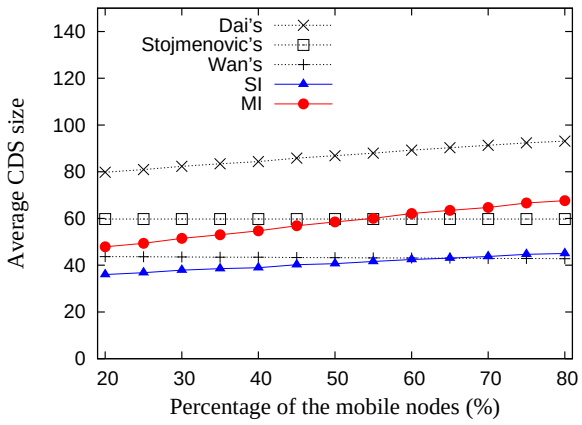

Figure 3.10: Average CDS size.

Figure 3.11 presents the number of extra messages to maintain CDS with respect to the percentage of the mobile nodes. As illustrated in Figure 3.11, MI requires a low number of extra messages to maintain CDS. The number of messages is only 20% of that of Wan's and 8% of Dai's. This demonstrates that MI can handle topology changes efficiently with only a small number of messages.

Figure 3.12 shows the average traffic required at each node to maintain CDS with respect to the percentage of the mobile nodes. As can be seen in Figure 3.12, the average traffic of MI and SI are at least 50% lower than that of Dai's and Stojmenovic's. Compared with Wan's and SI, MI has slightly higher traffic, but the difference is not significant. This is because the beacon in MI is slightly larger than that of Wan's and SI. As we have pointed out in Section 3.4.2, the traffic is mostly dominated by the extra bits in the beacon frame rather than the extra messages. This again is validated by Figure 3.11 and Figure 3.12.

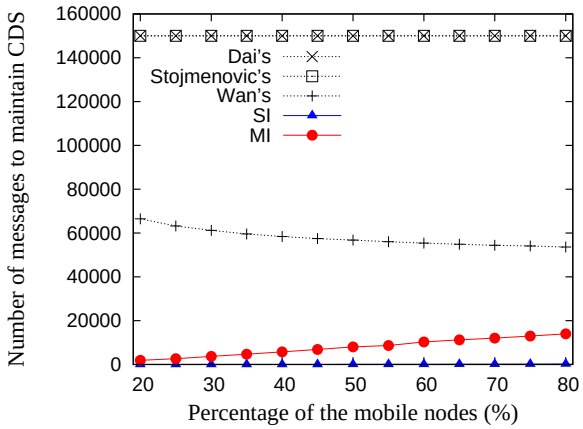

Figure 3.11: Number of messages to maintain CDS.

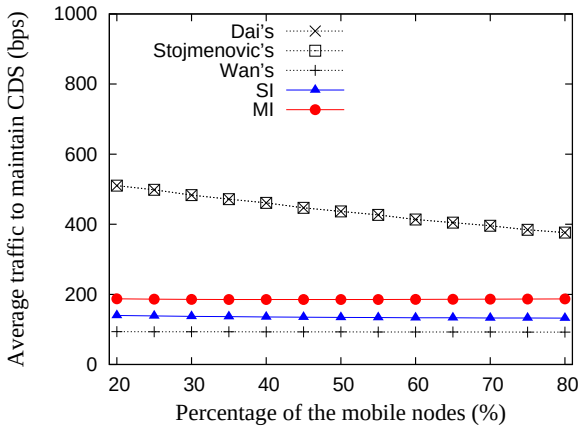

Figure 3.12: Average traffic to maintain CDS.

Figure 3.13 depicts the packet delivery rate with respect to the node density. In the case of sparse networks, AODV with MI results in lower delivery rate than the original AODV, AODV with Dai's, and AODV with Stojmenovic's. This is because in such low node density the network is often not connected, which leads to the failure of route discovery. However, when the average number of neighbors is more than 15, AODV with MI results in higher delivery rate than AODV, and has the similar delivery rate with AODV with Dai's and AODV with Stojmenovic's.

Figure 3.14 presents the end-to-end delay with respect to the node density. As can be seen in Figure 3.14, the delay of AODV increases in proportion to the number of neighbors, while that of AODV with MI remains stable. Compared with AODV with Dai's and AODV with Stojmenovic's, the delay of AODV with MI is only 50%. It implies that in networks with high density AODV, and AODV with Dai's and AODV with Stojmenovic's incur more collisions and take longer time to discover a route.

Figure 3.15 shows the number of RREQ packets with respect to the node density. It is clearly shown that AODV with MI introduces lower number of RREQ packets. This is because that only nodes in the CDS forward RREQ packets. On the other hand, AODV with Stojmenovic's incurs more number of RREQ packets than AODV with MI due to lower disconnections of a CDS. Considering the results presented in Figure 3.13, 3.14, and 3.15, we

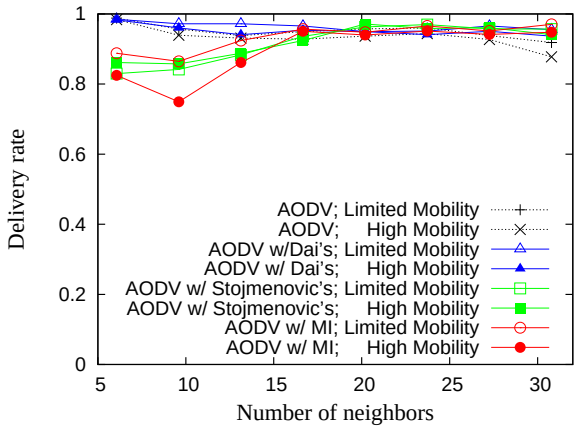

Figure 3.13: Delivery rate.

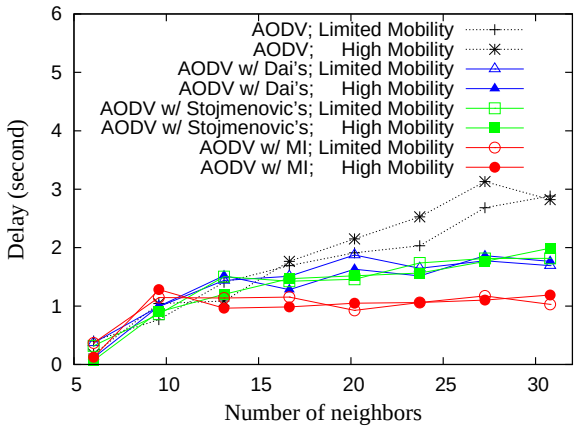

Figure 3.14: End-to-end delay.

are confident that CDS constructed by our proposed protocols improve routing performance and reduce routing overhead in MANETs.

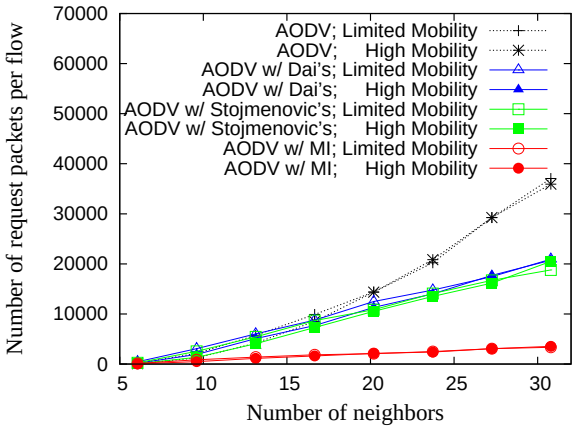

Figure 3.15: Number of RREQ packets.

3.5 Analysis

In this section, an analytical model to analyze the convergence time and number of messages that SI and MI require during CDS construction is presented and validated. Our analytical models estimate the average performance when nodes are randomly placed in a network in the uniformly distribution.

3.5.1 Analytical Model

The initiator election phase of SI and MI completes in $2Init_{Max}$ and α beacon periods, respectively. We first consider the convergence time of SI.

The convergence time of SI in the tree construction phase is the product of the number of hops from the initiator to the edge of the tree and the average defer time. To formulate this, let us denote the width and height of the simulation region as l_x and l_y, respectively. The number of hops, denoted by h, will be the distance from the initiator to the edge of the tree divided by the average distance between two nodes. Assuming $l_x = l_y$, and $S = l_x \cdot l_y$, then the average value of h can be computed by Equation 3.6.

$$h = \frac{\sqrt{l_x^2 + l_y^2}}{2\overline{d}} = \frac{3\sqrt{2S}}{4r} \tag{3.6}$$

The average defer time of a node is T_{max} divided by the average number of uncovered neighbors n_{uc}. Let C_A and C_B be the coverage area of two neighboring nodes A and B, and d be the distance between them, the additional area that B forwards a message from node A is $|C_B \setminus C_A| = \pi r^2 - |INTC(r,d)|$, where $INTC(r,d)$ is the intersection of two circles of radius r with their centers separated by d.

$$INTC(r,d) = 4\int_{d/2}^{r} \sqrt{r^2 - x^2}dx \tag{3.7}$$

Thus, the average additional coverage area is

$$\int_0^r \frac{2\pi x(\pi r^2 - INTC(r,x))}{\pi r^2}dx \approx 0.41\pi r^2 \tag{3.8}$$

Therefore, the average number of uncovered nodes is 0.41Δ, where Δ is the average number of neighbors. In addition, a node at the edge of the tree will wait T_{max} number of beacon intervals before it changes its *state* to *dominatee*. Finally, the convergence time of SI is approximately

$$2Init_{Max} + (h-1) \cdot \frac{T_{max}}{0.41\Delta} + T_{max} \tag{3.9}$$

The duration of the tree construction phase in MI can be calculated in the similar fashion. In the case of multi-initiator, the number of hops from an initiator to the edge of its tree, h_{mi}, is

$$h_{mi} = \frac{r/\bar{d}}{n_{init}\pi r^2/S} = \frac{3}{2} \cdot \frac{S}{n_{init}\pi r^2} \tag{3.10}$$

For the tree connection phase in MI, a border node without any uncovered neighbor will wait T_{max} number of beacon intervals before it sends a message to its initiator. The time required for the tree connection phase is bounded by $2h_{mi} + 1$. Hence, the convergence time of MI is the total time spent on the initiator election phase, the tree construction phase, and the tree connection phase, and can be computed as

$$\alpha + (h_{mi} - 1) \cdot \frac{T_{max}}{0.41\Delta} + T_{max} + 2h_{mi} + 1 \tag{3.11}$$

MI does not introduce extra messages except in the tree connection phase. In the tree connection phase, all nodes except initiators forward messages from their children as many times as the number of neighboring trees to their initiator. Thus, the number of message required for MI to construct CDS is

$$0.41\Delta \cdot \frac{n_{init} - 1}{n_{init}} \cdot n \tag{3.12}$$

3.5.2 Theoretical and Simulation Result Comparisons

In this subsection, the analytical model are validated by comparing with our simulation results.

Figure 3.16 demonstrates the convergence time of SI and MI with respect to the average number of neighbors. The convergence time decreases in proportion to the number of

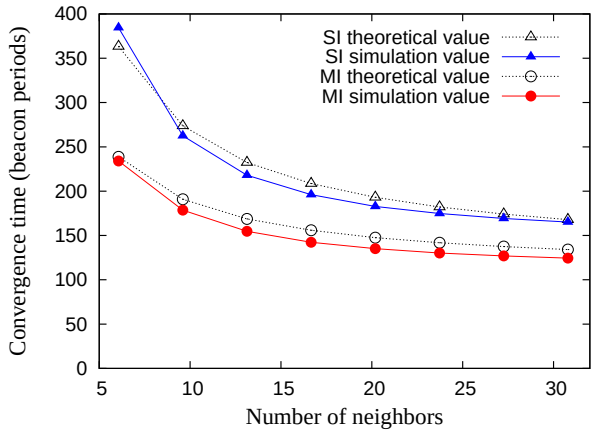

Figure 3.16: Convergence time.

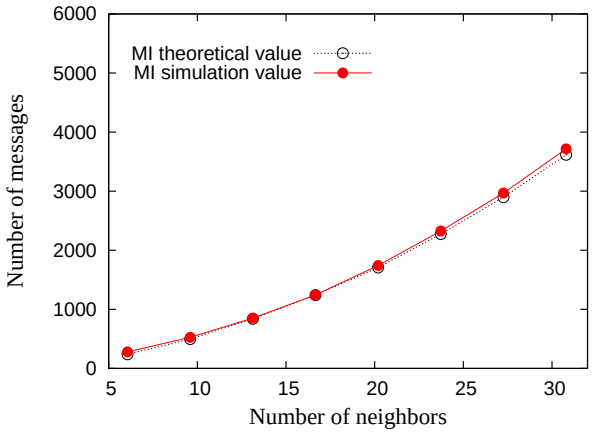

Figure 3.17: Number of messages.

neighbors because the defer timer in the tree construction phase decreases in proportion to the number of uncovered nodes. As can be seen in Figure 3.16, the theoretical model and simulation results are very close to each other.

Figure 3.17 shows the number of messages with respect to the average number of neighbors. In the simulation, the control messages in the tree connection phase are traced. Note that since SI forms only one dominator tree, there will be no control messages for the tree connection. As can be seen in Figure 3.17, analytical model again provides a very accurate estimation.

Chapter 4

Extensions of Tree-Based CDS Protocols

In this chapter, we introduces two optimization techniques to improve the performance of the tree-based protocols proposed in Chapter 3. To reduce the convergence time of the SI and MI protocols, Fast-Convergence Dominator Tree Construction (FC-DTC) is proposed in Section 4.1. For efficient mobility handling, Extended Mobility Handling (EMH) is introduced in Section 4.2

4.1 Fast-Convergence Dominator Tree Construction

4.1.1 The Issue of the Tree Construction

As described in Chapter 3, the tree-based CDS protocols are suitable for MANETs as they tend to result in smaller CDS, introduce less communication overhead, and adapt to nodal mobility. However, they still suffer from slow convergence due to the use of defer timers in their tree construction phase. In the rest of this chapter, the protocol used in the tree construction phase in SI and MI is referred as Timer-Based Dominator Tree Construction (TB-DTC). In TB-DTC, when a node finds that one of its neighbors joins the CDS, it sets a defer timer inversely proportional to the number of uncovered neighbors and joins the CDS only if it still has at least one uncovered neighbor when its timer expires. Recall that the defer timer is calculated using Equation 3.1.

Since nodes with more uncovered neighbors clearly have a shorter timer, they have a higher probability that at least one of its neighbors will still be uncovered when its timer expires, hence a higher probability to be included in the CDS. The convergence time of dominator tree construction is bounded by $h \cdot T_{max}$, where h is the number of hops from an

initiator to the edge of its dominator tree. When the network contains some long chains of nodes, it will take a long time for the protocol to converge. For instance, Figure 4.1 shows the worst scenario of TB-DTC. In Figure 4.1, the shaded square represents an initiator, a shaded circle represents a dominator, an unshaded circle represents a dominatee, and a dotted circle represents a covered or uncovered node. An arrow represents the relationship between the dominator and its dominatee (e.g., node 1 is the dominator of node 2), and a dotted line represents a link between two nodes. Suppose that in the initiator election phase, node 1 is elected as the initiator. Then, node 1 switches its *state* to *dominator* and covers node 2. On receiving beacons from its neighbors, node 2 learns that its only uncovered neighbor is node 3. Assuming $T_{max} = 100$ beacon periods, node 2 sets its timer, T_d, to 100 beacon periods. This process is repeated until all nodes turn to either *dominator* or *dominatee* as illustrated from the top to the bottom in Figure 4.1. The convergence time will be 300 beacon periods for this small chain with only 4 nodes. When the network density is low, this phenomenon will happen frequently, which will result in long CDS convergence time for TB-DTC.

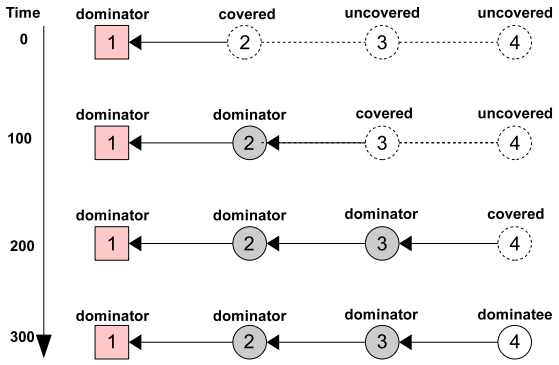

Figure 4.1: An example of a tree construction.

Another problem of TB-DTC is its poor scalability. In Equation 3.1, T_d has an upper bound of T_{max}. This is because in case of extremely high network density TB-DTC can not distinguish which node has more uncovered neighbors if each node has more than T_{max} number of neighbors.

The intuitive solution to slow convergence times is to use a different defer timer function, such as a log timer. If the log timer is upper bounded by a smaller value, the tree construction phase can be completed more quickly. However, this change will make the TB-DTC protocol even less scalable as more nodes will be mapped to the same defer timer value. Hence, it is preferable that the fast tree construction protocol does not rely on defer timers.

Therefore, in this section, we proposed a novel dominator tree construction protocol, namely Fast Convergence Dominator Tree Construction (FC-DTC) protocol. By using FC-DTC instead of TB-DTC, the convergence time of the SI and MI CDS protocols can be significantly improved. In the next subsection, we elaborate on the FC-DTC protocol.

4.1.2 Fast Tree Construction Algorithm

In addition to Table 3.1, the notations used in this chapter are given in Table 4.1. Just like TB-DTC, in FC-DTC each node encodes its *initiator* id, *state*, and *dominator* id into the beacon frame. In addition, each node also encodes the number of its uncovered nodes into its beacon. A node can be in either *dominator*, *dominatee*, *covered*, or *uncovered* state. At the beginning, all nodes are *uncovered*. Similar to what is defined in the IEEE 802.11 standard [25], each node periodically broadcasts its beacon. Through the exchange of beacons, a node learns the information of its one-hop neighbors, such as their *state* and n_{uc}.

Table 4.1: Definition of notations for FC-DTC and EMH.

Symbol	Definition
$N(i)$	The set of one-hop neighbors of node i
$N[i]$	$N(i) \cup \{i\}$
$flag(i)$	Set to TRUE if node i is a candidate dominator
$depth(i)$	The number of hops from $initiator(i)$ to i

After the initiator election phase, each initiator immediately switches its *state* to *dominator* and sets its initiator to itself. When an uncovered node receives the first beacon from a dominator neighbor, it sets its dominator to this neighbor, its initiator to the neighbor's initiator, and then switches its *state* to *covered*. By listening to beacons, each node knows how many uncovered neighbors it has. A boolean variable $flag$ is used at each node to track if a covered node should be switched to *dominator* state. If a node has the highest value of n_{uc} among its one-hop neighbors, which are in *covered* state, and belongs to the same dominator tree for at least two beacon periods, it sets its $flag$ to be 1 and waits for one beacon period. In the next beacon period, if it still has the highest value of n_{uc} and it has at least one uncovered node, it switches to *dominator* state. Otherwise, it sets its $flag$ back to 0 and stays in *covered* state. In other words, if a node has the highest value of n_{uc} for two beacon periods, it changes its state to *dominator*. If two nodes have the same value of n_{uc}, the tie can be broken by their *id*. During the process, if a node does not have any uncovered neighbor, it switches to *dominatee* state. Eventually, all nodes will switch their *state* to either *dominator* or *dominatee*. The pseudo code of this procedure is given in Algorithm 4.

FC-DTC constructs a dominator tree much quicker than TB-DTC. For instance, in Figure 4.1, after node 1 is elected as the initiator, in one beacon period node 2 will know it has the largest n_{uc} from the beacons. It then waits one additional beacon period and switches to *dominator* state. This process is repeated until all nodes are either *dominator* or *dominatee*, which will take only 7 beacon periods.

FC-DTC achieves the following two design goals. First, no matter how many neighbors each node has, FC-DTC guarantees that the node with the most uncovered nodes among one-hop neighbors becomes a *dominator* earlier. This will reduce the size of the CDS. Second, although FC-DTC extends the beacon frame by 8 bits, it still uses only one-hop localized information and does not introduce any extra control messages. In short, FC-DTC reduces the convergence time of CDS construction while at the same time keeping the advantages of TB-DTC.

Algorithm 4 FC-DTC pseudo code

```
/* Node i executes following */
Initialization:
    initiator(i) ← −1
    state(i) ← uncovered
    doinator(i) ← −1
    n_uc(i) ← −1
    flag(i) ← 0
Node i detects itself as initiator:
    initiator(i) ← i
    state(i) ← dominator
On receiving beacon from dominator node j:
if initiator(i) = −1 then
        initiator(i) ← init(j)
        dominator(i) ← j
        state(i) ← covered
end if
Node i in covered state ∧flag(i) = 0:
if i = candidate(i, initiator(i), N(i)) then
        flag(i) ← 1
        wait one beacon period
end if
Node i in covered state ∧flag(i) = 1:
if i = candidate(i, initiator(i), N(i)) then
    if ∃j ∈ N(i) ∧ state(j) = uncovered then
            state(i) ← dominator
    else
            state(i) ← dominatee
    end if
else
        flag(i) ← 0
end if
/* return id with the highest value of n_nc among N[i] */
Procedure candidate(i, init(i), N(i)):
    MaxID ← i
    MaxValue ← n_uc(i)
for all j in N(i) do
    if initiator(j) = initiator(i) ∧ state(j) = covered ∧ MaxValue < n_uc(j) then
            MaxID ← j, MaxValue ← n_uc(j)
    end if
    if state(j) = covered ∧ initiator(j) = initiator(i) ∧ MaxValue = n_uc(j) ∧ id(j) < id(i) then
            MaxID ← j, MaxValue ← n_uc(j)
    end if
end for
    return MaxID
```

4.1.3 Mobility Handling

Since the initiator election and the tree connection phases are not changed, the SI and MI protocols that incorporate our FC-DTC protocol can deal with nodal mobility in the

similar manner as the original SI and MI protocols. According to Section 3.2.4 and 3.3.4, the corruption of CDS is caused by one or more of the following five different nodal mobility cases.

1. The initiator leaves the network.
2. A new node joins the network after the construction of the CDS.
3. A redundant dominator node switches to the *dominatee* state while still keeping the dominating set connected.
4. A dominator node leaves the network.
5. A bridge node leaves its dominator tree.

It is clear that Case 1, 3, and 5 can be handled in exactly the same manner as the original SI and MI protocols. As discussed in Section 3.2.4, Case 4 can be handled by the color scheme because the color scheme does not rely on timers. Therefore, the only case that needs to be addressed is when a new node joins a dominator tree after the tree is constructed.

When a new node, say node j, joins a dominator tree, all its neighbors are in either *dominator* or *dominatee* state. If node j can find at least one dominator neighbor, it can be covered by one of its dominator neighbors. If node j cannot find a dominator neighbor, all the dominatee neighbors of node j's can switch to *covered* state. Then, the dominator tree construction phase begins again for node j and its neighbors. The pseudo code for handling this mobility case is given in Algorithm 5.

Algorithm 5 Additional pseudo code for mobility handling

```
1: /* Node i executes following */
2: Node i in dominatee state:
3: if ∃j ∈ N(i) ∧ state(j) = uncovered then
4:       state(i) ← covered
5:       flag(i) ← 0
6: end if
```

4.2 Extended Mobility Handling

4.2.1 The Issue of the Mobility Handling

While SI and MI can successfully adapt to topology changes, they suffer from two issues. First, the CDS recovery time may increase after a period of time. It is because when new nodes join the network, the height of the dominator tree may increase. When a bridge node moves away from the network, in the worst case, the initiator of one of the disconnected trees needs to assign a new bridge node by exchanging control messages between the initiator and nodes in the boundary area. The time to complete this recovery process is in proportion to the height of the tree. Note that during the recovery process, the dominating set will not be connected. Hence, to ensure the CDS to be available for a longer period of time, it is important that the height of the dominator tree to be small.

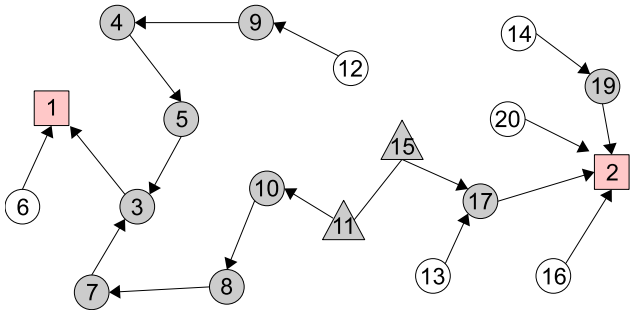

Figure 4.2: Original Topology I.

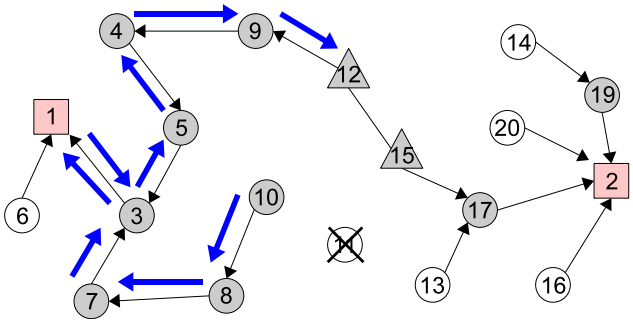

Figure 4.3: Mobility handling of MI on Topology I.

For instance, in Figure 4.2, a snapshot of a MANET and its connected dominating set are illustrated. In the figure, a square represents an initiator, a shaded circle is a dominator, a circle is a dominatee, and a triangle is a bridge node. The CDS is formed by the initiators, the dominators, and the bridge nodes. An arrow between two nodes indicates their dominator/dominatee relationship (e.g., node 3 is the dominator of node 5) and a solid line between two bridge nodes means they are neighbors to each other. Assume that the bridge node 11 runs out of power, according to MI, its dominator, node 10, will first look for an alternative bridge to connect to the neighboring tree. When node 10 fails to find such a node, it sends

a control message to its initiator, node 1. Then, node 1 designates a new bridge node, node 12, to connect to the neighboring tree. The control message will traverse 9 hops before a new bridge node can connect to the neighboring tree. This message traversal is illustrated in Figure 4.3.

Another issue of SI and MI mobility handling is the possibility of excessive initiators. When a connected network component breaks into multiple pieces due to mobility, each piece will find at least one initiator. When these pieces are merged, all these initiators will become part of the CDS. In addition, since each initiator generates a dominator tree, more initiators imply more trees, thus more bridge nodes will be needed to connect these trees. This further increases the size of CDS.

For example, in Figure 4.4, the topology is separated into three pieces, and each piece has its own initiator. After the change of topology, as shown in Figure 4.5, initiator 2 and 3 move to the proximity of the dominator tree rooted at initiator 1. To connect all these dominator trees, the MI protocol will include node 9, 10, and 14 into CDS.

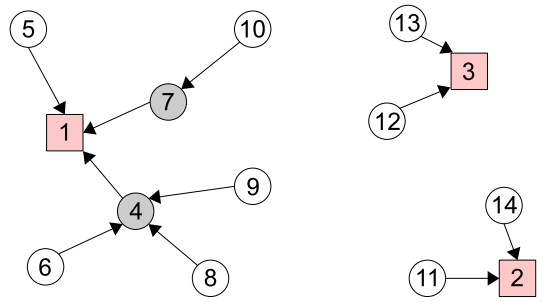

Figure 4.4: Original Topology II.

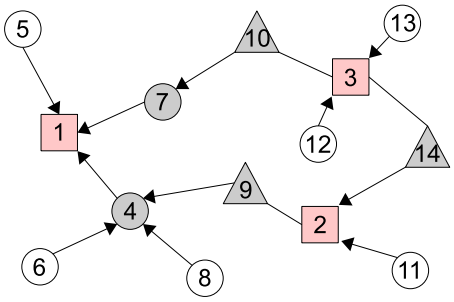

Figure 4.5: Mobility handling of MI on Topology II.

4.2.2 Extended Mobility Handling Algorithms

To address the aforementioned issues, we propose the Extended Mobility Handling (EMH) algorithm on top of SI and MI. EMH incorporates two procedures, namely Height-Reduction (HR) and Initiator-Reduction (IR). The following subsections elaborate on each of these procedures.

Height-Reduction

To control the height of a dominator tree, a node in the tree needs to know its depth, i.e., the minimum number of hops between its initiator and itself. Recall that in both SI and MI to learn the *state* of the neighboring node, the *state* of a node is encoded in the beacon. By adding an extra *depth* field in the beacon, the *depth* of a node can be obtained without introducing extra messages. When a node receives a beacon from its dominator, it learns and updates its own *depth*, then encodes its *depth* plus one to the *depth* field of its beacon before transmitting it. If a node finds a dominator neighbor which belongs to the same dominator tree and the *depth* of the dominator is smaller than its current dominator, it changes its dominator and updates its *depth* to reduce the height of the tree. The new dominator is able to know that the node becomes its child in the next beacon period. A dominator node which child changes its dominator may no longer have children. If this situation occurs, the dominator changes it *state* to *dominatee*. The pseudo code of the Height-Reduction procedure is given in Algorithm 6.

Algorithm 6 Height-Reduction

```
1: /* Node i executes following */
2: On receiving a beacon from j
3: /* changes its dominator */
4: if state(j) = dominator ∧ depth(j) < depth(domintor(i)) then
5:     dominator(i) ← j
6:     depth(i) ← depth(j) + 1
7: end if
8: /* eliminate unnecessary dominator */
9: if state(i) = dominator then
10:    if ∀j ∈ N(i), dominator(j) ≠ i then
11:        state(i) ← dominatee
12:    end if
13: end if
```

The following demonstrates how Height-Reduction works for the topology in Figure 4.2. Assume dominator node 3 is also a direct neighbor of node 8. After receiving beacons from node 3, node 8 knows $depth(3) = 1$ and $depth(7) = 2$. As node 3 is a dominator and closer to the initiator, node 8 changes its dominator to node 3 and updates its *depth*. Then, since dominator node 7 no longer has children, it changes its *state* to *dominatee*. Similarly,

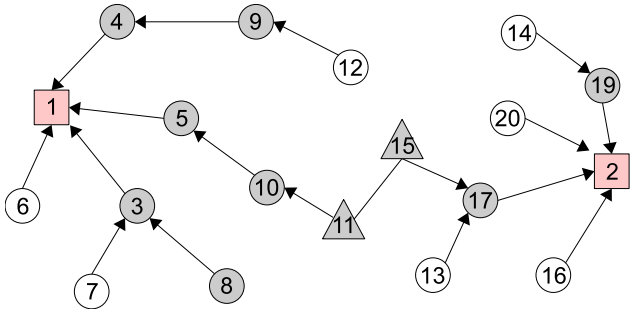

Figure 4.6: Topology with Height-Reduction

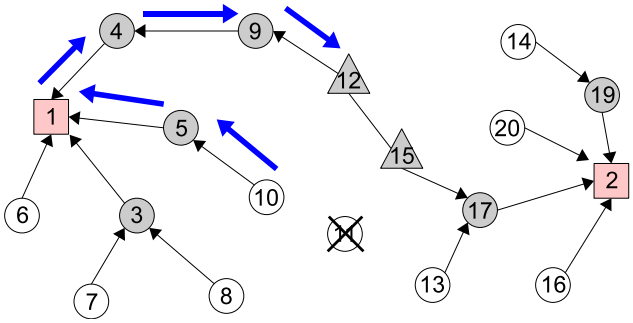

Figure 4.7: Mobility handling of MI after Height-Reduction

assume dominator node 5 is a direct neighbor of node 10. After node 10 finds that node 5 has a smaller depth, according to Height-Reduction it will switch its dominator to node 5 and update its *depth*. After a few beacon periods, the original dominator tree rooted at node 1 in Figure 4.2 will be optimized as shown in Figure 4.6. Now again assume that the bridge node 11 in Figure 4.6 runs out of power. As illustrated in Figure 4.7, the loss of the bridge node is handled in the same manner as in Figure 4.3. As the height of the tree in Figure 4.7 is smaller than that of Figure 4.3, the mobility handling for bridge node 11 can be done more efficiently. While MI needs to send 9 messages to recover the CDS, MI with Height-Reduction needs only 5.

Initiator-Reduction

Since each initiator generates a tree, reducing the number of initiators is equivalent to reducing the number of trees. To reduce the number of trees, the small dominator trees can be merged to the large neighboring trees. In essence, if a tree is small, most likely its initiator will also be a boundary node, i.e., the initiator has some neighbor belonging to a different tree. If an initiator finds such a neighbor and its *id* is larger than the initiator *id* of the neighbor, it changes its *state* to *dominator*, its initiator *id*, and updates its *depth*. The reason to merge the tree with larger initiator *id* to the one with smaller initiator *id* is to avoid the situation that both initiators of neighboring trees try to merge to the other tree simultaneously. After a dominator tree becomes a part of another tree, the children of the

merged initiator need to change their *state*. On receiving beacon from its dominator, if a node detects its dominator's initiator *id* is different from its initiator *id*, it updates its initiator *id* and *depth* accordingly. Notice that after Initiator-Reduction procedure is completed, an initiator will not have any neighbor belonging to a different tree. This guarantees that the tree of each initiator including the bridge nodes will be at least two hops in height. The pseudo code of the Initiator-Reduction procedure is provided in Algorithm 7.

Algorithm 7 Height-Reduction

```
1: /* Node i executes following */
2: /* merge small dominator trees */
3: if Node i is an initiator then
4:     dominator(i) ← j
5:     initiator(i) ← initiator(j)
6:     depth(i) ← depth(j) + 1
7:     if state(j) = dominatee then
8:         state(j) ← dominator
9:     end if
10: end if
11: /* On receiving a beacon from j (change initiator id) */
12: if dominator(i) = j ∧ initiator(j) ≠ initiator(i) then
13:     initiator(i) ← initiator(j)
14:     depth(i) ← depth(j) + 1
15: end if
```

Figure 4.8 shows how Initiator-Reduction works for the topology in Figure 4.4. Recall that in Figure 4.5, initiator 2 and 3 move to the boundary area. Node 2 is still an initiator, and it has two neighbors 9 and 12 that associate with other initiators. According to the pseudo code, node 2 will change its initiator *id* and update its *depth*. Afterwards, node 9 finds that node 2 become its child, so it is no longer a bridge node. When Node 11 and 14 find that their dominator changed its initiator *id*, they will update their initiator *id* and *depth*. As can be seen, the size of CDS by the MI protocol's mobility handling in Figure 4.5 is 8, while by MI with Initiator-Reduction in Figure 4.8 is reduced to 7.

4.3 Simulations

To evaluate the performance of SI with FC-DTC, MI with FC-DTC, SI with EMH, and MI with EMH, we implement our protocols in C++ along with the other CDS protocols,

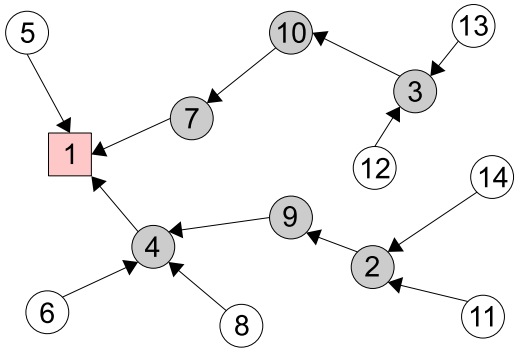

Figure 4.8: Topology with Initiator-Reduction.

including Dai's [7], Wan's [9], the original SI, and the original MI protocols. Notice that the original SI and MI protocols use TB-DTC in the dominator tree construction phase. We do not include Stojmenovic's protocol [8], as it has similar results to Dai's. In this section, the simulation results of different CDS protocols are reported and analyzed.

4.3.1 Simulation Configurations

The performance of FC-DTC and EMH is assessed in the static network scenario and mobile network scenario, respectively. The simulation configurations and evaluation metrics are the same as Section 3.4.1. Hence, we only describe the configurations of FC-DTC and EMH. For FC-DTC, in addition to the original SI and MI, both SI with FC-DTC and MI with FC-DTC enlarge the beacon by adding n_{uc} (8 bits per beacon). For EMH, the only one difference is in the beacon frame format. To implement EMH in SI and MI, we extend the beacon of SI and MI by adding *depth* (4 bits per beacon).

4.3.2 Simulation Results for Static Network Scenario

Figure 4.9 shows the CDS size of different protocols with respect to the network density for different protocols. It is clear that SI with FC-DTC consistently generates the smallest CDS and Dai's consistently generates the largest CDS among all the protocols. In addition, except Dai's protocol, the size of CDS generated by other protocols is not sensitive to the

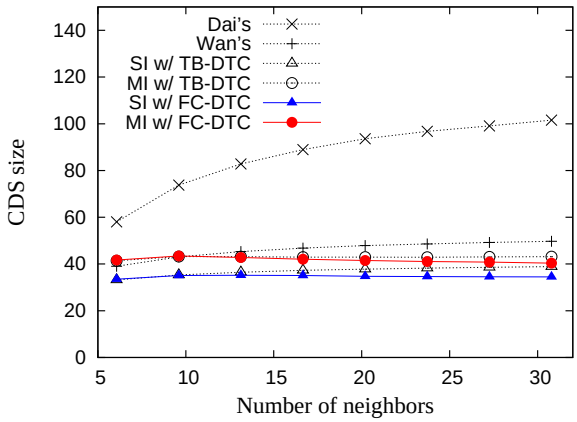

Figure 4.9: CDS size.

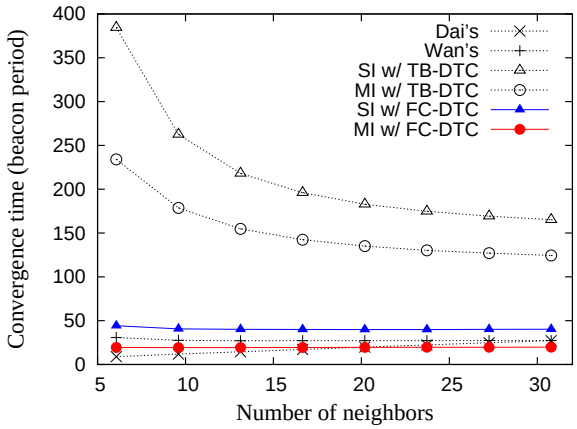

Figure 4.10: Convergence time.

density of the network. In the case of high network density, SI with FC-DTC and MI with FC-DTC result in a slightly smaller size for CDS than the original SI and MI protocols. This is because FC-DTC guarantees that nodes with more uncovered neighbors will be switched to *dominator* first. This suggests that FC-DTC is more scalable than TB-DTC. While it has been proven that the size of Wan's CDS is suboptimal [9], both SI with FC-DTC and MI with FC-DTC are able to generate CDS with a smaller size.

Figure 4.10 presents the convergence time with respect to the network density for different protocols. As shown in Figure 4.10, SI with FC-DTC and MI with FC-DTC significantly reduce the convergence time compared with the original SI and MI protocols. Even at the high network density, SI with FC-DTC and MI with FC-DTC reduce the convergence time by more than 80% compared with their respective TB-DTC counterpart. While the convergence time of the original SI and MI protocols are affected by the network density, that of SI with FC-DTC and MI with FC-DTC is stable. Notice that MI with FC-DTC creates CDS faster than even Dai's protocol. Compared with Wan's protocol, the convergence time of SI with FC-DTC and MI with FC-DTC is almost the same.

Figure 4.11 demonstrates the number of extra messages with respect to the network density for different protocols. As illustrated in Figure 4.11, MI with FC-DTC introduces only 30% of the extra messages that Dai's does. Compared with Wan's, MI with FC-DTC

reduces the number of extra messages up to 60% when the network density ranges from 5 to 15 neighbors per node. Note that both SI with TB-DTC and SI with FC-DTC does not introduce any extra message since they do not have the tree connection phase.

Figure 4.12 shows the average traffic incurred by the CDS construction with respect to the network density for different protocols. Obviously, Dai's protocol introduces the largest traffic, and the traffic increases in proportion to the network density. This is because of the inclusion of the neighboring list in the beacon frame. For the other five protocols, the average traffic is almost the same. Note that SI with FC-DTC and MI with FC-DTC enlarge the beacon frame of the original SI and MI protocols by only 8 bit. It is interesting that even though the number of messages for MI with FC-DTC, MI with TB-DTC, and Wan's protocols increase in proportion to the number of neighbors as shown in Figure 4.11, the traffic remains relatively constant to the network density. This implies that the traffic is mostly dominated by the extra bits in the beacon frame.

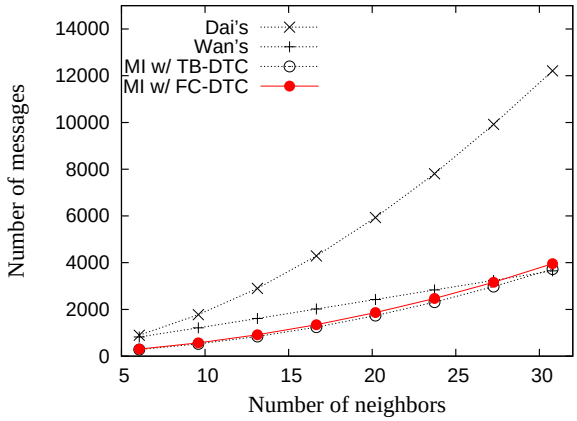

Figure 4.11: Number of messages.

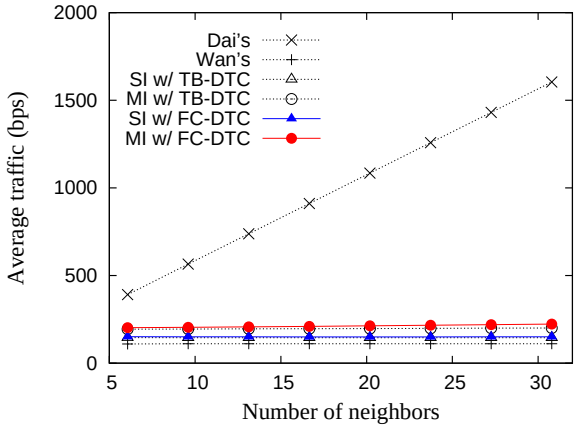

Figure 4.12: Average traffic (bps).

4.3.3 Simulation Results for Mobile Network Scenario

Figure 4.13 illustrates the percentage of time CDS is alive with respect to the percentage of the mobile nodes. As can be seen in Figure 4.13, MI with EMH almost always has the highest percentage of CDS alive time than other CDS protocols. Although smaller CDS

is generally more vulnerable to topology changes, MI with EMH shows excellent mobility adaptation compared with the other protocols. Only when the percentage of mobile nodes is greater than 70% the percentage of CDS alive time of the MI-EMH becomes slightly lower than that of Dai's due to MI with EMH's smaller CDS size. As pointed out in [9], the time complexity of mobility recovery at each node of Dai's is as high as $O(\Delta^2)$, where Δ is the average number of neighbors. Thus, Dai's protocol takes more time to recover than MI with EMH. SI with EMH also improves the percentage of CDS alive time by at least 10% compared with SI. This clearly demonstrates that EMH is capable of prolonging the time CDS is available to MANETs.

Figure 4.14 shows the average CDS size with respect to the percentage of the mobile nodes. As illustrated in Figure 4.14, SI and SI with EMH consistently produce the smallest CDS and Dai's protocol consistently produces the largest CDS. While the size of CDS generated by MI increases slowly in accordance with the percentage of mobile nodes, that of MI with EMH is stable. This is because, by merging trees MI with EMH keeps the number of initiators as a constant, and consequently reduce the CDS size. In general, it is desirable that the size of the CDS is as small as possible for MANETs. From Figure 4.13 and Figure 4.14, MI with EMH can quickly adapt to topology changes while keeping CDS size competitive.

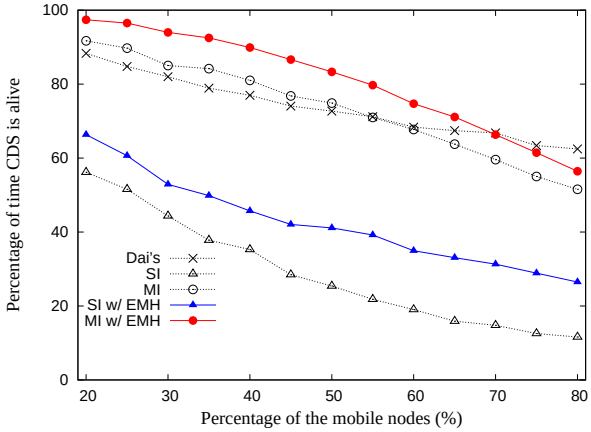

Figure 4.13: Percentage of time CDS is alive.

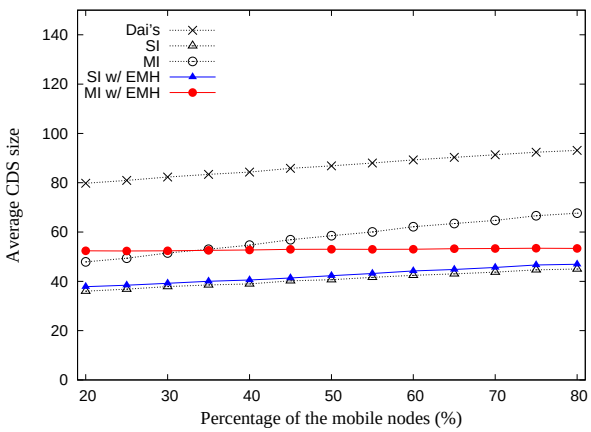

Figure 4.14: Average CDS size.

Figure 4.15 presents the number of extra messages to maintain CDS with respect to the percentage of the mobile nodes. As illustrated in Figure 4.15, SI does not introduce any extra message. MI with EMH introduces less than 25% of extra messages of Dai's. In addition, the numbers of extra messages of MI and SI with EMH are roughly the same and are at most 10% of that of Dai's. Compared with MI, MI with EMH introduces more messages. The same can be said to SI and SI with EMH. Unlike the other protocols, the number of extra message created by MI with EMH decreases as the percentage of the mobile nodes increases. This is because IR can better reduce the number of initiators when more initiators move to the boundary area. By introducing few extra control messages, MI with EMH and SI with EMH become highly adaptive to topology changes, as shown in Figure 4.13.

Figure 4.16 shows the average traffic required at each node to maintain CDS with respect to the percentage of the mobile nodes. As can be seen in Figure 4.16, the average traffic of Dai's protocol is at least twice as much as that of any other protocol. The protocol incorporates EMH has slightly higher traffic than the one without EMH, but the difference is not significant. This is primarily because in EMH the beacon is 4 bit larger to include the depth value. Figure 4.15 and Figure 4.16 suggest that the traffic is mostly dominated by the extra bits in the beacon frame.

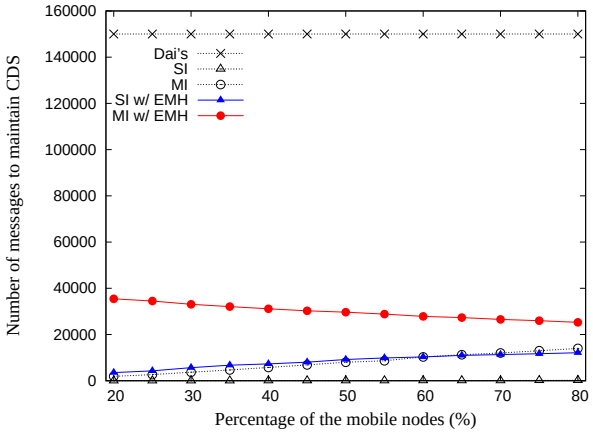

Figure 4.15: Number of messages to maintain CDS.

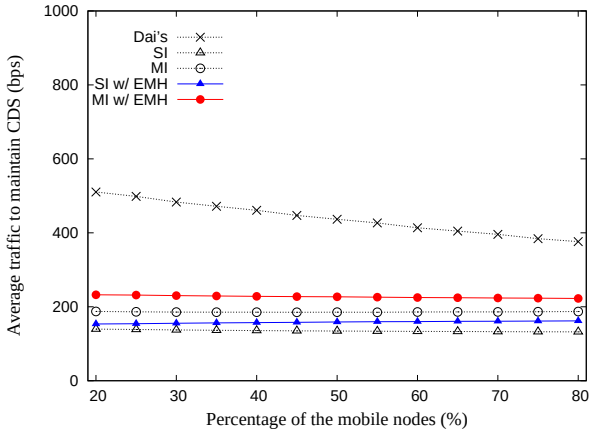

Figure 4.16: Average traffic to maintain CDS.

4.4 Analytical Model for The Convergence Time

In this section, an analytical model to analyze the convergence time that SI with FC-DTC and MI with FC-DTC require during CDS construction is presented and validated.

The one-hop neighbors of an initiator requires one beacon period to become dominator or dominatee, as they have enough information to initiate the protocol during the initiator election phase. Afterwards, it takes two beacon periods to advance one hop from the initiator. Thus, the convergence time in the dominating tree construction of FC-DTC can be calculated by $2 \cdot (h-1)+1$ for SI and $2 \cdot (h_{mi}-1)+1$ for MI, respectively.

In the initiator election phase, it takes at least α beacon periods to elect a α-hop local minimum. For the original SI protocol, α is set to be 20, so $h = \frac{3\sqrt{2S}}{4r}$, where S is the simulation region and r is the transmission range. Hence, SI with FC-DTC, the convergence time is obtained by Equation 4.1.

$$\alpha + 2(h-1) + 1 = \alpha + 2h - 1 \tag{4.1}$$

As described in Section 3.5, the convergence time in the tree connection phase for the original MI protocol is $2h_{mi}+1$. And, α is set to be 2, so $h_{mi} = \frac{3}{2} \cdot \frac{S}{n_{init}\pi r^2}$. Here, n_{init} is the number of local minimum. When $\alpha = 2$, n_{init} is approximately $\frac{9}{25} \cdot \frac{S}{\pi r^2}$. Therefore, MI with FC-DTC, the convergence time is estimated by Equation 4.2.

$$\alpha + 2(h_{mi}-1) + 1 + 2h_{mi} + 1 = \alpha + 4h_{mi} \tag{4.2}$$

Figure 4.17 depicts the values calculated from our analytical model and the results obtained from the simulation. As can be seen in Figure 4.17, our analytical model provides a very accurate estimation in terms of the convergence time for SI with FC-DTC and MI with FC-DTC.

Remark: Note that so far we have assumed a node can collect the updated information from all its neighbors within one beacon period. In reality, since the transmission of beacons is

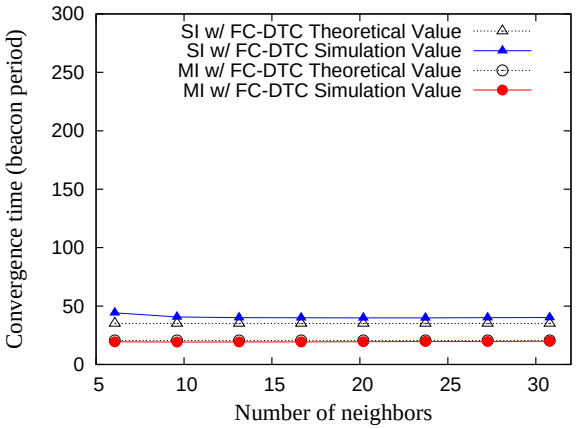

Figure 4.17: Convergence time.

generally done without coordination, there may be collisions between beacon transmissions. To accommodate this issue, each node can wait for a fixed number of beacon periods, say T_{wait}, after it finds it has the highest value of n_{uc}. In this case, the convergence time of the dominator tree construction for SI with FC-DTC and MI with FC-DTC will be $(T_{wait}+1)h_{mi}$ and $(T_{wait}+1)(h_{mi}-1)+1$, respectively. This increases the convergence time slightly, but the result will still be much faster than SI with TB-DTC and MI with TB-DTC. For instance, when $T_{wait} = 3$, the convergence time is less than 50 beacon periods for SI with FC-DTC and less than 33 beacon periods for MI with FC-DTC. Even with the highest network density (30 neighbors per node), the convergence time is at least 165 beacon periods for SI with TB-DTC and at least 124 for MI with TB-DTC.

Chapter 5

Conclusions

While the existing CDS protocols are successful in constructing a CDS of small size, they miss a number of key features that are important in mobile ad hoc networks. In this paper, we introduce two tree-based CDS protocols, namely Single-Initiator (SI) and Multi-Initiator (MI), that not only create a CDS of competitive size with low overhead but also address the shortcomings of the existing protocols. SI utilizes timers to distributively construct and maintains CDS in the presence of changes of network topology. Built on top of SI, MI requires minimum localized information to construct and maintain CDS efficiently. The performance of the SI and MI protocols are verified by C++ and ns-2 simulations under static/mobile network settings, and an analytical model. Both performance assessments provide very close results, which establishes the validity of our simulations.

In addition to the two tree-based CDS protocols, we introduced two extensions to improve the performance of SI and MI, namely Fast-Convergence Dominator Tree Construction (FC-DTC) and Extended Mobility Handling (EMH). FC-DTC reduces the convergence time required in constructing a dominator tree by avoiding the use of differ timers. On the other hand, EMH successfully reduces the recovery time and keeps a CDS competitive size by controlling the size of dominator trees and the number of initiators during mobility process. Simulation results show that SI and MI incorporating FC-DTC and EMH significantly improve the performance of the original SI and MI protocols.

Bibliography

[1] J. Blum, M. Ding, A. Thaeler, and X. Cheng, "Connected Dominating Set in Sensor Networks and MANETs," *Handbook of Combinatorial Optimization*, pp. 329–369, 2005.

[2] J. Wu, "Extended Dominating-Set-Based Routing in Ad Hoc Wireless Networks with Unidirectional Links," *IEEE Transaction on Parallel Distributed Systems*, vol. 13, no. 9, pp. 866–881, 2002.

[3] J. Cartigny, D. Simplot, and I. Stojmenovic, "Localized Minimum-Energy Broadcasting in Ad-hoc Networks," in *Proceedings IEEE Conference on Computer Communications (INFOCOM)*, Mar. 2003, pp. 2210–2217.

[4] A. Helmy, S. Garg, P. Pamu, and N. Nahata, "CARD: A Contact-based Architecture for Resource Discovery in Ad Hoc Networks," *ACM Baltzer Mobile Networks and Applications (MONET)*, vol. 10, no. 1, pp. 99–113, 2004.

[5] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*, 1990.

[6] J. Wu and H. Li, "On Calculating Connected Dominating Set for Efficient Routing in Ad Hoc Wireless Networks," in *Proceedings of the 3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM)*, Aug. 1999, pp. 7–14.

[7] F. Dai and J. Wu, "An Extended Localized Algorithm for Connected Dominating Set Formation in Ad Hoc Wireless Networks," *IEEE Transaction on Parallel Distributed Systems*, vol. 15, no. 10, pp. 908–920, 2004.

[8] D. Simplot-Ryl, I. Stojmenovic, and J. Wu, "Energy-Efficient Backbone Construction, Broadcasting, and Area Coverage in Sensor Networks," *Handbook of Sensor Networks: Algorithms and Archtectures*, pp. 343–379, 2006.

[9] P. J. Wang, K. M. Alzoubi, and O. Frieder, "Distributed Construction of Connected Dominating Set in Wireless Ad Hoc Networks," in *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, Apr. 2002, pp. 141–149.

[10] K. M. Alzoubi, P.-J. Wan, and O. Frieder, "Message-Optimal Connected Dominating Sets in Mobile Ad Hoc Networks," in *Proceedings of the 3rd ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, Jun. 2002, pp. 157–164.

[11] P.-J. Wan, L. Wang, and F. Yao, "Two-Phased Approximation Algorithms for Minimum CDS in Wireless Ad Hoc Networks," in *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, Jun. 2008, pp. 337–344.

[12] X. Cheng, M. Ding, D. H. Du, and X. Jia, "Virtual backbone construction in multihop ad hoc wireless networks: Research articles," *Wireless Communications and Mobile Compututing*, vol. 6, no. 2, pp. 183–190, 2006.

[13] Y. Li, M. T. Thai, F. Wang, C.-W. Yi, P.-J. Wan, and D.-Z. Du, "On greedy construction of connected dominating sets in wireless networks," *Wireless Communicationss and Mobile Compututing*, vol. 5, no. 8, pp. 927–932, 2005.

[14] J. Wu, F. Dai, M. Gao, and I. Stojmenovic, "On Calculating Power-Aware Connected Dominating Sets for Efficient Routing in Ad Hoc Wireless Networks," *IEEE/KICS Journal of Communications and Networks*, vol. 4, no. 1, pp. 59–70, 2002.

[15] B. Kim, J. Yang, D. Zhou, and M.-T. Sun, "Energy-Aware Connected Dominating Set Construction in Mobile Ad Hoc Networks," in *Proceedings of the 14th IEEE International Conference on Computer Communications and Networks*, Oct. 2005, pp. 229–234.

[16] F. Dai and J. Wu, "On Constructing k-Connected k-Dominating Set in Wireless Networks," in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2005.

[17] Y. Wu, , F. Wang, M. T. Thai, and Y. Li, "Constructing Algorithms for k-Connected m-Dominating Sets in Wireless Sensor Networks," in *Proceedings of the IEEE Military Communications Conference (MILCOM)*, Oct. 2007, pp. 29–31.

[18] Y. Wu and Y. Li, "Construction Algorithms for k-Connected m-Dominating Sets in Wireless Sensor Networks," in *Proceedings of the 9th ACM International Symposium on Mobile Ad Hoc Networking and Computing (Mobihoc)*, May 2008.

[19] S. Yang, J. Wu, and F. Dai, "Efficient Directional Network Backbone Construction in Mobile Ad Hoc Networks," *IEEE Transaction on Parallel Distributed Systems*, vol. 19, no. 12, pp. 1601–1613, 2008.

[20] D. S. Hochbaum, Ed., *Approximation Algorithms for NP-hard Problems*. PWS Publishing Co., 1997.

[21] S. Guha and S. Khuller, "Approximation Algorithms for Connected Dominating Sets," *Algorithmica*, vol. 20, no. 4, pp. 374–387, 1998.

[22] B. S. Baker, "Approximation algorithms for NP-complete problems on planar graphs," *J. the ACM*, vol. 41, no. 1, pp. 153–180, 1994.

[23] S. Basagni, M. Mastrogiovanni, A. Panconesi, and C. Petrioli, "Localized Protocols for Ad Hoc Clustering and Backbone Formation: A Performance Comparison," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 4, pp. 292–306, 2006.

[24] S. Basagni, M. Mastrogiovanni, and C. Petrioli, "A Performance Comparison of Protocols for Clustering and Backbone Formation in Large Scale Ad Hoc Networks," in *Proceedings IEEE Conference on Mobile Ad-hoc and Sensor Systems*, Oct. 2004, pp. 70–79.

[25] "IEEE 802.11b Standard," http://standards.ieee.org/getieee802/download/802.11b-1999.pdf.

[26] D. Zhou, M.-T. Sun, and T. Lai, "A Timer-based Protocol for Connected Dominating Set Construction in IEEE 802.11 Multihop Mobile Ad Hoc Networks," in *Proceedings of International Symposium on Applications and the Internet (SAINT)*, Jan. 2005, pp. 2–8.

[27] "The Network Simulator (ns-2)," http://www.isi.edu/nsnam/ns/.

[28] W.-J. Hsu, K. Merchant, H.-W. S., C.-H. Hsu, and A. Helmy, "Weighted Waypoint Mobility Model and its Impact on Ad Hoc Networks," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 9, no. 1, pp. 59–63, 2005.

[29] "Mobilab: Community-Wide Library of Mobility and Wireless Networks Measurements," http://nile.usc.edu/MobiLib/.