Implementing a Connected Dominating Set Algorithm for Clustering Mobile Ad Hoc Networks

ABSTRACT

Advances in network technology have tremendously changed the way people communicate. The relatively recent introduction of wireless networking technology has the potential to play an even more influential role in our daily lives. However, the nature of wireless technology makes it a more difficult platform on which to build useful systems. Some particularly troubling aspects of wireless networks include high mobility, frequent failures, limited power, and low bandwidth. A core component of a networked system is its routing protocol. Several ad-hoc wireless routing protocols have been proposed which depend on a flood-and-cache design. However, these algorithms suffer from scalability problems whenever the hit rate in the cache is low, such as when most connections are short-lived. This paper describes the design and implementation of a Deployable Connected Dominating Set (DCDS) algorithm. As in other CDS algorithms, DCDS provides a scalable routing scheme by constructing and maintaining a backbone across the network. To make our CDS algorithm truly deployable in an IEEE 802.11 network, we eliminate three unrealistic assumptions on which previous designs are based: reliable broadcast, accurate neighbouring information, and a static setup phase. We have implemented the DCDS algorithm and simulated it using Glomosim. The evaluations show that DCDS has significantly better scalability than AODV. We also show that our algorithm can maintain an effective backbone which appropriately balances setup time and size.

1. INTRODUCTION

Wireless networking is emerging as one of the hottest new commodities of the early part of the twenty-first century. Personal computing is increasingly shifting from desktops to laptops and from Ethernet to IEEE 802.11 [1] wireless local-area networks. PDAs and cell phones are merging to provide computing, web-access, email and instant messaging along with standard voice telephony. The developed world may soon be at the point where most people carry a wireless-networking device virtually everywhere they go.

At present, communication among wireless devices is typically based on a wired infrastructure. IEEE 802.11, the standard for laptop wireless networking, for example, utilizes wired access points to connect wireless devices. Access points organize nearby wireless devices and handle all of their communication. Digital cellular technology utilizes a similar technique for PDA and cellphone based Internet access. A graph of these common infrastructure-based networks is thus comprised of traditional wired links, with only the leaf nodes connected wirelessly.

This infrastructure approach has many advantages, but it has two main weaknesses. First, a wireless network can only exist in places where wired access points have been established. Second, even where this infrastructure is in place, it is vulnerable to being overloaded if too many wireless devices are operating within its radio range. While the first problem may improve over time, as wireless networking becomes more popular, the second is likely to get worse as the density of wireless devices increases.

The existing networking technology offers a tantalizing, but largely unutilized alternative. These technologies allow nodes to self organize to form ad-hoc networks without the aid of any wired infrastructure (e.g., IEEE 802.11, for example, defines both Infrastructure and Ad Hoc modes). Such a network could provide local-area communication to support service discovery, instant messaging, game playing and other activities among a geographically co-resident set of nodes. It could also extend the range, but not the bandwidth, of a wired infrastructure, by providing multiple wireless hops to reach a wired node. Finally, it could increase the aggregate bandwidth among densely populated regions of wireless devices by allowing devices to attenuate transmission power to reduce interference and then connecting the emasculated devices by an ad hoc network.

This paper is concerned with the development of an effective protocol for unicast and multicast communication in 802.11 mobile, ad-hoc wireless networks. We show through simulation that previous solutions suffer severe performance problems due to network congestion. One class of algorithms that includes AODV [2,3], DSR [4,5], and DSDV [6] uses a combination of broadcast flooding and route caching to discover network routes among wireless nodes. These algorithms perform poorly when route information is not cached or when cached information is inaccurate due to failure or node mobility. Another class of algorithms that includes CDS [7–21] address this shortcoming by proactively maintaining certain routing information, in the form of a spanning graph. These algorithms make unrealistic assumptions about the underlying network and are thus not deployable in an 802.11 environment.

The fundamental challenge involves management of the shared airspace around each wireless device. While operating in infrastructure mode, the access point arbitrates the airspace using a polling scheme and explicit acknowledgement MAC-level messages. Wireless nodes send messages only when granted clearance and re-send if their messages are not acknowledged by the access point. While some of this support can be used in adhoc mode, several challenges arise.

First, every protocol relies on flooding to discover routes in the wireless network. The most efficient way to implement this flooding is to use *broadcast* messages that are received and processed by every in-range node. Despite that 802.11 *unicast* messages deploy a *request-to-send/clear-to-send/data/ack* procedure, broadcast messages, however, are sent without receiving a clear-to-send signal and without acknowledgement. They are thus considerably less reliable than unicast messages. Nevertheless, the correctness of current CDS protocols rest on reliable broadcast. Furthermore, injudicious use of flooding can clog the shared airspace, creating message storms that render the network temporarily useless [22]. Our simulations show that it is precisely this issue that causes considerable problems for the flood-and-cache algorithms such as AODV.

Another problem occurs when routing messages hop-by-hop across the network. In this case the more reliable unicast messages are used, but the allocation of the shared medium can not be granted centrally as it is in infrastructure mode. As a result, even unicast messages are more vulnerable to congestion loss in adhoc mode. The problem is exacerbated considerably if the distance between hops is small compared to each node's radio range. In this case, multiple hops of the same message share the same airspace, thus increasing congestion. Our simulations show that these two issues can cause significant, even debilitating message loss for existing algorithms.

We present the design and simulation of a new CDS algorithm based on Alzoubi and Frieder's CDS algorithm [10]. Our design improves on all earlier algorithms by removing three key limitations. First, these algorithms use broadcast messages to set up a spanning graph, assuming erroneously that broadcast messages can be delivered reliably in a wireless network. We tolerate message loss. Second, these algorithms erroneously assume that a wireless node can accurately know the identities of all other nodes in its radio range (i.e., its neighbours). We allow this information to be only approximately correct. Third, previous algorithms assume that a stop-the-world setup phase can establish the graph with only minor adaptations subsequently being required to maintain connectivity. Our approach is robust towards node mobility and failure. It builds the graph incrementally as nodes come and go, ensuring that the graph at any point in time is as completely connected as it would have been in the original CDS algorithm immediately following the end of the setup phase.

The rest of the paper is organized as follows. We briefly survey related work in the area in Section 2. In section 3, we briefly describe the Message-Optimal CDS algorithm and its problems that we are addressing. Then, we detail our DCDS algorithm and current routing scheme in Section 4 and 5 respectively. In Section 6, we provide an initial performance characterization of our new algorithm using simulation. Some ideas for future work are mentioned in Section 7, and finally a conclusion is made in Section 8.

2. RELATED WORK

There are many MAC/PHY protocols that have been proposed to support wireless communication in mobile ad hoc networks (MANETs), among which IEEE 802.11 is the most widely accepted. Because a wireless node uses the shared airspace for its transmission, avoiding signal interference and message collision is inherently difficult. To alleviate these problems, IEEE 802.11 provides a Carrier Sense Multiple Access / Collision Avoidance (CSMA/CA) scheme. This scheme asks each node to check the medium activity and backoff accordingly before it transmits any packets. Further, it specifies a Request-To-Send (RTS) / Clear-To-Send (CTS) / Data / Ack procedure for sending each unicast packet. The exchange of RTS and CTS messages not only acts as a fast collision/interference check, but also enables the sender to reserve the shared medium for its usage. If a sender does not receive its receiver's CTS or ACK message, it will retry the transmission until it receives the correlated ACK or until a retry limit is reached. Therefore, utilizing such a procedure enables IEEE 802.11 to provide a reliable unicast function. However, the broadcast function can not take advantage of this scheme because a broadcast packet may be intended for any number of receivers and the RTS/CTS/DATA/ACK scheme does not easily generalize. Therefore, the reliability of broadcast is reduced due to the increased probability of lost frames from interferences, collisions, or time-varying channel properties [1].

Although wireless nodes can talk to each other within a one-hop distance using MAC layer protocols, they require routing algorithms to accomplish end-to-end communication across multiple hops. We can divide the existing mobile, ad-hoc routing algorithms into two classes based on how they create and manage routing information. The first class, which includes AODV, DSR, and DSDV, generally uses a flood-and-cache approach to discover and maintain each individual route across the network. These algorithms can be further subdivided based on how they handle topology changes: proactively or reactively. Proactive algorithms such as DSDV require nodes to periodically flood routing updates to the rest of the network. This approach has two main problems. First, route updates are sent to every node, though only a subset may need this information. Second, the repair of a broken route is delayed until the receipt of the next update message; packets sent along the broken route in the meantime are lost. Reactive algorithms such as AODV, on the other hand, fix broken routes on demand as they are detected. Once a reactive algorithm sets up a link, no further broadcast is required until a link on the route breaks. Broch et al. [23] demonstrate that reactive algorithms generally outperform proactive ones in terms of delivery ratio and routing overhead for long-lived end-to-end connections. Both proactive and reactive approaches, however, rely on broadcast floods to establish and repair connections one at a time and thus have heavy overhead for short-lived connections or dynamic networks.

The second class of routing algorithms reduces broadcast overhead by basing routes on a shared-backbone spanning graph. Instead of setting up and maintaining a separate route for each source-destination pair, these algorithms select a subset of nodes, called the Connected Dominating Set (CDS), to organize a backbone across the network. The resulting route aggregation can significantly lower overhead by reducing the total number of routing nodes and links. The benefits of this approach increase with network density and mobility.

An intuitive way to obtain a CDS is to first identify a dominating set (DS), and then grow it into a CDS by adding connector vertices. A DS is a subset of nodes that is able to reach all the nodes the network. Gerla et al. [12, 18] present distributed algorithms that follow this approach by using lowest node ID or highest node degree to pick nodes into the dominating set. These results are generalized by Basagni [13] to use an abstract weight to determine membership. Das et al. [14, 17, 19] decentralize Guha and Khuller's algorithm [20] and come up with a series of algorithms for setting up a network backbone. Wan et al. [8] point out, however, that these algorithms require global synchronization and suffer from exponential time and message complexity.

Chen and Liestman formalize the idea of a weakly-connected dominating set (WCDS) [9, 11], where two dominators can be separated by a two-hop distance. Based on Guha and Khuller's centralized spanning tree algorithm [20], their greedy approximation algorithms can generate a WCDS for a static ad hoc network. Although they propose distributed versions of the centralized algorithms, these algorithms still require a node (the tree root) to have the accurate topology information of its own network partition [11], and to control the growth of the spanning tree.

Wu and Li [7] present a localized distributed CDS algorithm. In contrast to previous methods, their algorithm first generates a big CDS, and then removes the redundant vertices to reduce the size of the CDS. To complete their algorithm, Wu et al. [16, 21] propose a scheme to address topology changes in mobile networks. Even though these algorithms are simple and intuitive, they are not able to generate a small-size CDS with a constant approximation ratio [8]. Furthermore, the mobility scheme requires nodes to detect mobility and keep an up-to-date list of two-hop neighbours, which makes it hard to deploy.

Alzoubi, Wan and Frieder [8, 10, 15] propose a series of distributed algorithms for finding small connected dominating sets. Compared to previous algorithms, the approximation ratios of these algorithms are bounded by constants. The Message-Optimal CDS algorithm [10] is purely localized and is able to achieve both linear message complexity and linear time complexity. They also apply the CDS algorithm locally within a three-hop distance to address network topology changes.

3. BACKGROUND

Our CDS algorithm is based on the Message-Optional Connected Dominating Set (MOCDS) algorithm [10]. Before describing our algorithm it is thus useful to provide background on this previous algorithm and to point out the weaknesses in it that we address.

3.1 Overview of Message-Optional CDS

A Dominating Set (DS) of a graph $G(V, E)$ is a vertex subset V' of V, such that each node in V is either in V' or adjacent to a vertex in V'. Two nodes are adjacent when they are in radio range of each other; we also say that these nodes are within *one hop* of each other. If the set V' is a connected subgraph, it is called a Connected Dominating Set (CDS). When using the CDS as a routing backbone, the smaller the number of CDS nodes is, the more benefit this routing scheme can provide. It has been proven, however, that finding a Minimum Connected Dominating Set (MCDS) is an NP-Complete problem [24]. Previous CDS research has thus focused on heuristics finding a small CDS. The quality of such a graph is measured by the ratio of the graph size it produces compared to the theoretical Minimal CDS for a given topology. This ratio is referred to as the algorithm's *approximation ratio*.

The Message-Optimal CDS algorithm (MOCDS) is the only distributed and localized algorithm we are aware of that constructs a CDS with a constant approximation ratio and linear time and message complexity [10].

There are four types of nodes in the MOCDS algorithm: *candidates*, *dominators*, *connectors* and *dominatees*. Each node starts as a candidate. The MOCDS algorithm selects a subset of nodes, the dominators, to comprise a Maximum Independent Set (MIS). Dominators are chosen such that every node is within one hop of a dominator, but no two dominators are within one hop of each other. Each dominator is assigned a set of connector nodes that connect it to every other dominator within its three-hop radio range. The dominators and connectors together constitute the CDS backbone. All other nodes are dominatees. Every node is assigned a unique ID and stores a complete ID list of its neighbours. To construct a CDS, MOCDS requires each node to maintain a local data structure that records the following information:

- *node ID*
- *state:* its current role
- *x1:* the number of its candidate neighbours
- *x2:* the number of its candidate neighbours with lower IDs
- *y:* the number of its neighbours that have reported their one-hop dominators
- *z:* the number of its neighbours that have not reported their one-hop and two-hop dominators
- *list1:* its one-hop neighbouring dominators
- *list2:* its two-hop neighbouring dominators as well as the next hop to reach each one (a dominator only records such information about its two-hop neighbouring dominators with higher IDs)
- *list3:* its three-hop neighbouring dominators with larger IDs and the next hop to reach each, only for a dominator
- *Clist:* its adjacent connectors
- *Rlist:* routing information, only for connectors

The MOCDS algorithm operates in two phases. It begins with an initial set-up phase in which the CDS backbone is established. Once the set-up is complete, the algorithm transitions to maintenance phase, in which the algorithm performs local backbone repairs when nodes fail or move. We now review the operation of the algorithm in these two phases.

3.1.1 Backbone Construction

Initially, nodes are assigned a state of candidate and have a complete neighbour list to decide the values for *x1* and *x2*, but no other information about the network.

To begin, each node examines its *x2* to determine whether its ID dominates all of its neighbours by being the smallest. If so, it changes its state to dominator and broadcast a DOMINATOR message that includes its ID.

When a node receives a DOMINATOR message, it adds the dominator to its *list1* and decrements *x1* by one. If it is still a candidate, the node changes its state to dominatee and broadcasts a DOMINATEE message.

When a candidate node receives a DOMINATEE message, it decrements its *x1* by one and checks if the sender has a lower ID. If so, it decrements its *x2* by one. Once all its neighbours with smaller IDs are dominatees, it becomes a dominator and broadcasts a DOMINATOR message.

When a node has received DOMINATOR and DOMINATEE messages from all its neighbours, it broadcasts a LIST1 message to inform its neighbours of the dominators within its range.

When a node receives a LIST1 message, it adds the dominators it has not already seen to its *list2*, along with the ID of the sender. It also updates its *y* variable to indicate that it has received a LIST1 message. When a node has received LIST1 messages from all of its non-dominator neighbours, it broadcasts its *list2* in a LIST2 message. A dominator node follows a similar procedure when receiving a LIST2 message to update its *list3*, and sends a LIST3 message when it has received LIST1 and LIST2 messages from all of its neighbours.

When a dominatee or connector receives a LIST3 message, it searches each *list3* path in the message for its own ID to determine whether it is a connector for the path. Each path contains two dominators and one or two other nodes that connect these dominators. If it finds such a path, the node changes its state to connector, and if necessary, adds each matching path in its *Rlist* and broadcasts a CONNECTOR1 message that contains its entire *Rlist*. When a node receives a CONNECTOR1 message for which it is also a connector, it adds the matching entries in the message to its *Rlist* and broadcasts a CONNECTOR2 message that contains its *Rlist*. When any node receives a CONNECTOR1 or CONNECTOR2 message it adds the sender to its *Clist*.

3.1.2 Backbone Maintenance

During maintenance phase, MOCDS performs local repair operations when necessary in an attempt to adapt the backbone when nodes fail or move. It does this by classifying every topology change as one of three cases: (1) dominator node movement, (2) dominatee/candidate node movement, and (3) connector node movement.

Generally speaking, the maintenance procedure consists of two steps. First, any node that detects a topology change collects local topology information from its neighbours. The node then applies the MOCDS construction algorithm locally within its three-hop neighbourhood. This reconstruction is triggered by special messages that indicate what sort of topology change has been detected. During the process of a local maintenance, MOCDS requires each involved node to have accurate local topology information in order to conduct the construction algorithm.

3.2 Problems of the MOCDS Algorithm

While MOCDS's formal properties make it the best of its class, four debilitating problems occur when attempting to deploy it.

First, the algorithm assumes that nodes initially have a complete and accurate list of their neighbours, so that nodes can track the states of their neighbours during the setup phase. When receiving a DOMINATEE message, for example, a node must know whether all of its dominating neighbours (i.e., those with smaller IDs) are dominatees. If so, the node becomes a dominator. Nodes also use the list in the initial step to determine whether to send a DOMINATOR message and in later steps to know when they have received LIST1 and LIST2 messages from all neighbours.

In a real network, however, nodes can only establish their neighbour lists by exchanging broadcast messages, which are prone to loss in congested networks. Furthermore, nodes may move or fail during the setup phase, making it virtually impossible to ensure that a neighbour list is accurate for the entire phase. As a consequence, in a typical network, some neighbour lists will miss listing some neighbours, leading to networks with too many dominators. Worse, some neighbour lists will list non-existent neighbours, leading to deadlock when expected messages never arrive.

The second problem is related to the first. Each of the control messages sent in MOCDS is broadcast and the algorithm assumes that each broadcast is received and processed by all in-range nodes. In reality, however, broadcast messages are not transmitted reliably in even moderately loaded networks. Thus, even if the neighbour lists are complete and accurate for the entire setup phase, the algorithm will typically still deadlock because some control messages are not received by every intended node.

The third problem is that the MOCDS algorithm makes a distinction between the set-up phase and the maintenance phase. In the set-up phase, it assumes that the network topology remains unchanged. Therefore, each node can be confident that its one-hop neighbours list will not change and the CDS construction algorithm will converge. In reality, the normal case for an MANET is not starting with a large number of nodes. Instead, the typical case is to add new nodes to an existing network. Moreover, the dynamic nature of an MANET determines that network changes and exceptions are inevitable. As a result, it is hard to draw a line between these two phases and provide MOCDS with a global snapshot in the construction process.

The fourth problem is that the MOCDS maintenance phase relies on the same assumptions as the construction phase: reliable broadcast, complete neighbourhood knowledge, and a static three-hop neighbourhood during the process of backbone maintenance. In addition, it does not provide a way to detect topology changes. Moreover, the maintenance algorithm is quite expensive. It needs synchronization and collaboration from all the nodes within a three-hop distance regardless of whether a topology change actually matters. Thus, it generates many redundant and unnecessary broadcast control messages.

4. DEPLOYABLE CDS ALGORITHM

We have developed a new algorithm, called Deployable CDS (DCDS), that modifies MOCDS to allow it to be deployed reliably in a real mobile, ad-hoc network while retaining its good formal properties. This section describes how DCDS differs from MOCDS and how it addresses the MOCDS weaknesses outlined earlier.

4.1 The Synopsis

There is only one phase, i.e. the maintenance phase, during the life time of a CDS backbone. Therefore, DCDS has only three node states: *dominator*, *dominatee* and *connector*. Each node starts as a dominator. Afterwards, it adjusts itself to its dynamic surroundings with a best effort approach. Hence, a static network environment, which is used for constructing a backbone, is no longer a must.

DCDS is a purely localized event-driven maintenance scheme, in which a node makes a decision based only on its current state and the packet it receives. Thus, it relaxes the assumption that each node should have a complete and accurate neighbour list. Further, a node no longer has to synchronize with all its neighbours when there is a topology change.

DCDS completes and facilitates the maintenance of the CDS backbone in mobile ad hoc networks. The detection scheme is a proactive approach that involves all the nodes, and the dominatees and connectors are responsible for maintaining the backbone connectivity. Utilizing a periodic one-hop broadcast message, a node is able to *conjecture* topology changes within its two-hop radio range, and only reacts to those important ones that affect the CDS routing function. In addition, except for the heartbeat messages and the CANCEL-CLUSTER-REPORT message, we replace all the broadcast messages with a set of unicast ones. This change does not impose significant overhead on each node because we adopt some optimizations to minimize the number of control messages. Instead, it improves the reliability and promptness of communications by alleviating the broadcast collision problem, and saves the power consumption for each node.[1]

[1]The IEEE 802.11 power-save scheme enables a node to stay in doze mode if it is not involved in a transmission.

Table 1: General Information of Control Messages

Message Name	Transmission Method	Sender Role	Functions
DOMINATOR	One hop broadcast	Dominator	Notify its neighbours about its existence; Provide its connectivity info
DOMINATEE	One hop broadcast	Dominatee or Connector	Notify its one-hop dominators about its existence; Provide its neighbouring dominators' connectivity info
NEW-CLUSTER-REPORT	Unicast	Dominatee or Connector	Report to a neighbouring dominator about possible new backbone connections
CANCEL-CLUSTER-REPORT	One hop broadcast	Dominator	Cancel further NEW-CLUSTER-REPORT about specified dominators
LOST-CLUSTER-REPORT	Unicast	Connector	Report to a neighbouring dominator about possible broken backbone connections
CONNECTOR1	Unicast	Dominator	Designate the first connector
CONNECTOR2	Unicast	Connector	Designate the second connector for a three-hop backbone connection
ACK	Unicast	Dominatee or Connector	Confirm a connection has been established

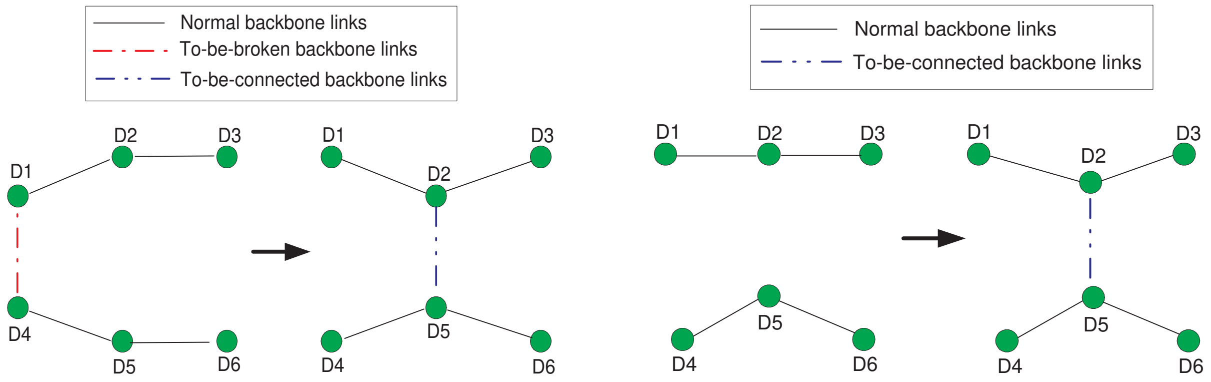
4.2 Local Structure and Control Messages

We follow three criteria to reduce the scope and complexity of the local structure maintained by each node. First, we preclude those variables that require accurate neighbours information, such as *x1*, *x2*, *y* and *z*. Second, we exclude those unnecessary variables from the local structure. We regard a variable as necessary if it contains routing information or is used to maintain the backbone connectivity. Thus, *Clist* is no longer needed. Third, we aggregate and simplify all the routing information, and each node maintains only part of the information according to the role it is playing. We keep the *list1* as *Dominators list*, and only the non-dominator nodes need to maintain it. To enable the routing function on the backbone, we combine *list2*, *list3* and *Rlist* and save the routing information in both the dominators and the connectors. Further, we require each dominator to have a *Dominatees list* so that it has a rough knowledge about its one-hop neighbours. We no longer require a non-dominator node to record two-hop neighbouring dominators in *list2*. The new local data structure is shown as follows:

- *node ID*
- *state*: its current role
- *Dominators list*: its one-hop neighbouring dominators, only for a non-dominator node
- *Dominator ID*: its backbone relay point, only for a non-dominator node
- *Dominatees list*: its one-hop neighbouring non-dominator nodes, only for a dominator
- *Dominator-connectivities info list*: the connectivity status of its one-hop neighbouring dominators, only for a non-dominator node
- *Routing table*: routing information, only for a backbone node

We point out that, all the information stored in the above data structures is not necessarily accurate and complete. Each node in an MANET makes decisions based upon this local knowledge from its own perspective. Although some of these decisions could be wrong, our algorithm will eventually fix them.

As mentioned earlier, the DCDS algorithm utilizes a different set of control messages. These control messages enable it to effectively retain backbone connectivity and minimize maintenance overhead. Further, by using these control messages, DCDS relaxes the need for global or neighbouring synchronization and allows each node to promptly react to any topology change on its own. We briefly list their related information in Table 1 and will elaborate their functions in the following sections.



(a) Case I

(b) Case II

Figure 1: Two Possible Topology Change Cases in an MANET

4.3 Heartbeat Messages

In a mobile wireless network, a CDS backbone faces many problems when the network topology changes. For example, in the case where a connector moves away and this causes a broken backbone link, the backbone needs to repair it with alternative connectors. The previous works that include the MOCDS algorithm use localized CDS maintenance schemes [10, 21], but they do not provide ways to detect topology changes or identify moving nodes. We were also attempting to make a local and reactive maintenance algorithm, which repairs a backbone connection only when a dominator or connector fails to send a data packet. However, we argue that, with real mobile settings, it is insufficient without support from external infrastructures, such as GPS. This is because of two reasons. First, it is hard for a mobile node to identify itself without a fixed reference point since node movement is relative to each other. Second, a reactive maintenance algorithm does not provide a dominator with the ability to detect new neighbouring clusters, although data delivery failures can be used to indicate a broken backbone connection. A local and reactive CDS repair depends on the assumption that all the other parts of a network are fully connected at any time. But this is not necessarily true given these two limitations.

Taking the first case shown in Figure 1(a) for example, $D1$ and $D4$ move away from each other and eventually, when they are separated by more than a three-hop distance, the local repair can no longer link them together. At the same time, the $D2$ and $D5$ are moving near to each other, and we can potentially create a new backbone connection between them. However, with a locally reactive maintenance algorithm, this potential backbone link can not be detected, and this ends up with two isolated backbones. For the same reason, we can see that in Figure 1(b), when two previously unconnected backbone pieces move towards each other, this maintenance scheme also fails to merge them together.

To address these problems, we come up with one proactive backbone maintenance algorithm, which requires participation from all the nodes in the network to detect topology changes. Namely, each node periodically broadcasts one-hop control messages to its neighbours. We call these one-hop periodic messages *Heartbeat Messages*. DCDS inherits the DOMINATOR and DOMINATEE messages from the MOCDS algorithm as the heartbeat messages. However, it modifies their structures so that they can piggyback the current backbone connectivity information. For example, when a dominator node sends a DOMINATOR message, it includes its current connected neighbouring dominators in the message. Likewise, a non-dominator node fills in a DOMINATEE message with the entries in its *Dominators list*.

Therefore, the heartbeat messages can provide the following functions. First, they enable a node to *speculate* the topology changes in its neighbourhood and react to them accordingly. Each node applies an individual timer for each entry in its *Dominators list*, *Dominatees list* and *Routing table*, and these heartbeat messages are responsible to refresh them. If any timer expires, a node believes that the corresponding neighbour has left and invalidates all the related routing information. Second, these heartbeat messages provide a dominator node with the ability to fully connect to any other neighbouring dominators within a three-hop distance. Every time a non-dominator node receives a heartbeat message, it learns the connectivity status of its neighbouring dominators, and thus potentially is able to create new backbone connections. Last, the heartbeat messages with timers together help each node to adapt itself to the dynamic surroundings individually without any synchronizations with its neighbours. A node assumes that the information it collects from the heartbeat messages is accurate enough and makes a decision only based on its current state and the ongoing event, such as a timeout or a packet reception. Although it is possible that a heartbeat message may contain inaccurate or incomplete topology information, or even cannot be delivered successfully, the successive heartbeats can eventually fix all the consequences.

4.4 Backbone Maintenance

The DCDS algorithm provides a maintenance scheme to keep a CDS backbone in good shape. If any of the following rules were broken, DCDS should be able to fix it automatically.

- Any node is either a dominator or adjacent to at least one of them.
- No two dominators can communicate with each other directly within a one-hop distance.
- Two neighbouring dominators are at most three hops away.
- The dominators are locally fully connected and are able to reach any nodes in the network.

The DCDS algorithm no longer requires the neighbours to synchronize with each other when there is a topology change. As mentioned earlier, utilizing the heartbeat messages and the timers, a node is able to perceive any change in its neighbourhood and react to it locally. Afterwards, it declares the decision to its neighbours through DOMINATOR or DOMINATEE heartbeat messages. Its neighbours may react to this change accordingly. For example, a dominator u moves adjacent to another dominator v and it has a lower *node ID*. Thus, v decides to change its state to dominatee and records u into its *Dominators list* once it receives u's DOMINATOR message. However, the dominatees and connectors within v's cluster are affected by this decision. Later, if any of them receives the following DOMINATEE heartbeat message from v or its timer for v expires, it deletes v from its *Dominators list* and selects another dominator if the list is not yet empty. Otherwise, it backs off for a random time. If it still can not discover any neighbouring dominator, it declares itself as a dominator and immediately sends a DOMINATOR heartbeat message notifying its neighbours about this decision. During the course of the maintenance, we can see that any involved node independently observes the network changes and then draws conclusions on its own.

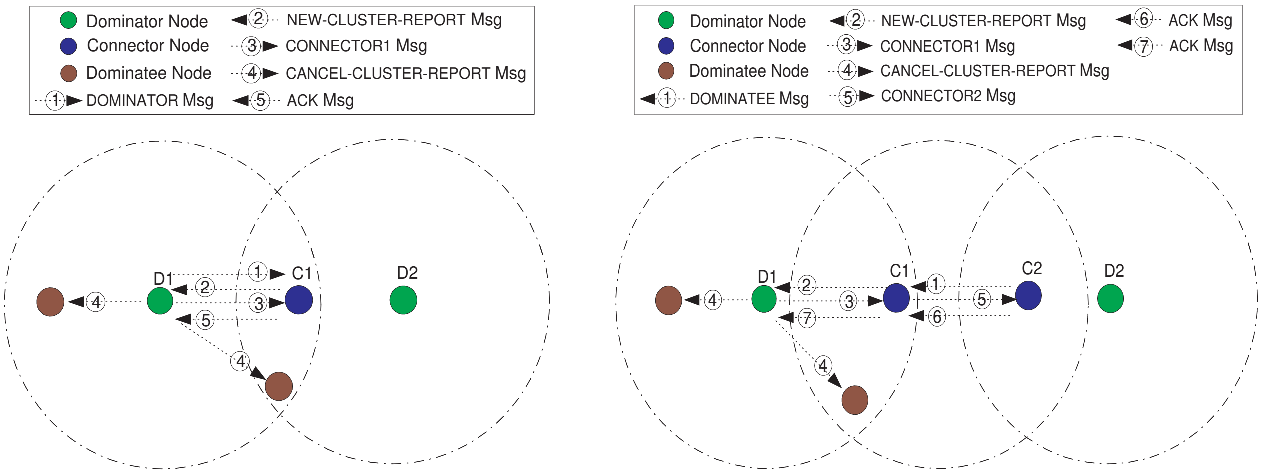
The DCDS algorithm uses a four-item tuple, *(D1, C1, C2, D2)*, to stand for a unidirectional connection. *D1* is the starting dominator. It can reach the other dominator, *D2*, through *C1* and *C2* consecutively. In a two-hop connection, *C2* must be null. These connection tuples are stored in the *Routing tables* of the dominators and the connectors, comprising a backbone across the network. To create a backbone connection, the DCDS algorithm adopts a similar scheme to the CDS construction algorithm used in the MOCDS. However, it also makes some modifications other than changing most of the broadcast control messages to unicast ones.

First, it designates the responsibility of discovering a new backbone connection to the dominatees and connectors, instead of the dominators. By doing this, it stops each dominatee or connector from periodically sending the unnecessary LIST2 messages, which are used for setting up three-hop connections. Instead in the DCDS algorithm, a dominatee or connector is able to detect a new backbone connection using the heartbeat messages, and sends a NEW-CLUSTER-REPORT message to inform the starting dominator only when it finds one.

Second, it introduces two new control messages: the ACK message and the CANCEL-CLUSTER-REPORT message. An ACK message is used to confirm that a backbone connection has been successfully established and is ready for use. Unless the starting dominator receives such a message, the new connection stays invalid and is not deployed for routing. A CANCEL-NEW-CLUSTER-REPORT message is a one-hop broadcast message sent by a dominator. It, together with a jitter, is able to prevent the *unicast storm* from happening, especially in a dense and mobile network. Before a dominatee or connector reports a new backbone connection to a neighbouring dominator, we ask it to buffer this report first for a random jittering time. If it receives such a CANCEL-NEW-CLUSTER-REPORT message about this connection, it simply cancels the report since that dominator has been already informed. Although setting each node into promiscuous mode can also help solve this problem, we do not use it for the reason of power consumption.

Figure 2(a) shows the process of creating a two-hop backbone connection. As mentioned before, *C1* maintains its neighbouring dominators in its *Dominators list*. It can also perceive *D1*'s current connectivity status when it receives a DOMINATOR heartbeat message from *D1*. Provided with this information, it first calculates those neighbouring dominators which *D1* has not yet connected. Then, it sends a NEW-CLUSTER-REPORT packet with these dominators' information to *D1*. In this example, this NEW-CLUSTER-REPORT message informs *D1* about *D2*'s existence. When *D1* receives such a message and if it has not yet responded to the same report sent from other neighbours, it sends back a CONNECTOR1 message asking *C1* to be a connector immediately followed by a CANCEL-CLUSTER-REPORT message. Getting such a request, *C1* changes its state to connector, inserts a connection tuple into its *Routing table* and sends back a positive ACK if *D2* is still in its *Dominators list*. Otherwise, it just ignores this request. Only after receiving the ACK, *D1* inserts such a connection into its *Routing table*, updates its connectivity information, and finally a two-hop backbone link is created.

Creating a three-hop backbone connection takes a similar approach as shown in Figure 2(b). Compared to cre-



(a) Creating a Two-hop Backbone Connection (b) Creating a Three-hop Backbone Connection

Figure 2: Building up Backbone Connections

ating two-hop connections, there are two major differences. First, creating a three-hop connection happens only when a dominatee or connector receives a DOMINATEE heartbeat message. Each DOMINATEE message contains information about the sender's neighbouring dominators. When *C1* receives such a message from *C2*, it notices that there is a new dominator *D2* which *D1* has not connected. Therefore, it generates a NEW-CLUSTER-REPORT message and sends it to *D1*. Second, *C1* has to send a CONNECTOR2 message to *C2* after it receives a CONNECTOR1 message. If *C2* agrees to be the second connector, it sends *C1* an ACK message. *C1* sends its ACK message back to *D1* only after obtaining this confirmation.

We proactively, as well as reactively, detect and remove obsolete connections. Any backbone node maintains two timers for every entry in its *Routing table*: a Previous-Hop timer (PH timer) and a Next-Hop timer (NH timer). The PH timers in a starting dominator must be null. Each of the timers is refreshed by related DOMINATOR or DOMINATEE heartbeat messages. Given a connection *(D1,C1,C2,D2)*, for instance, *D1*'s NH timer is reset by *C1*'s DOMINATEE heartbeat messages. *C1* or *C2*, if *C2* is not null, refreshes its PH or NH timer accordingly every time it receives a heartbeat message from its previous-hop or next-hop backbone node. We need to point out that the other dominator, *D2*, requires no timers for this connection because its *Routing table* does not include such a tuple. If any timeout happens, a backbone node believes that it has detected a broken backbone connection and generates a LOST-CLUSTER-REPORT message. It sends this message to its previous-hop backbone node, if it is not the starting dominator and this timeout originates from the NH timer. This message is relayed in the backbone until it reaches the starting dominator. When receiving this report, the starting dominator deletes the correlated connection from its *Routing table* and updates its connectivity status. In the connection example mentioned above, if *C2* is not null and it recognizes that *D2* has gone, it composes a LOST-CLUSTER-REPORT message and sends this message to *D1* through *C1*. After receiving this message, *D1* stops forwarding any further data packets through this broken backbone connection and removes it immediately from its routing table.

This proactive broken backbone detection algorithm is effective, nevertheless it has two major weaknesses. First, because it depends on timeouts, the proactive algorithm is relatively slow to react to the topology changes. Second, a LOST-CLUSTER-REPORT message can be dropped because of mobility, failure or congestion. As a result, it is possible that a starting dominator can never find an outdated backbone connection. For example, *C1* recognizes that *C2* has gone and sends such a report to *D1*. However, this unicast packet is dropped in the MAC layer. Afterwards, as long as *D1* and *C1* stay adjacent to each other, *D1* cannot find out that this connection is actually broken. This causes the following packets sent through this backbone connection to be dropped at *C1*. Therefore, we exploit another reactive broken backbone detection algorithm. This algorithm does not depend on those heartbeat messages and timers. Instead, a connection deletion occurs when a dominator or connector fails to send a data packet through a backbone link. When a packet drop happens, the connector(s) attempts to notify the starting dominator by sending back a LOST-CLUSTER-REPORT message. Obviously, it is able to solve the two problems of the proactive approach. However, in a mobile network where the number of data packet is not that significant, only deploying this reactive algorithm results in a poor-quality CDS backbone. Therefore, we adopt both of them in DCDS.

5. DCDS ROUTING ALGORITHM

At present, the DCDS algorithm uses a simple routing scheme. We divide the process of packet delivery into three steps. First, the source node sends a packet to its dominator. Then, the dominator relays it on the backbone in a *unicast flooding* manner. Finally, when any dominator locates the destination node, it forwards that packet to the destination, and a packet delivery succeeds. Otherwise, if the destination node cannot receive the packet for any reason, the packet delivery fails.

The *Dominator ID* and the *Dominatees list* facilitate the last-hop deliveries between the non-dominator nodes and the backbone, and the *Routing table* helps a dominator forward a packet to all its neighbouring dominators, except for the one that the packet comes from. In order to alleviate the congestion problem, each dominator records the packets it recently received. Whenever it receives a packet, it first checks whether it has seen this packet already. If so, it just drops this packet.

6. EVALUATION

The previous CDS works evaluate a CDS backbone by its approximation ratio and a good CDS algorithm should have its approximation ratio bounded by a constant. Since the DCDS algorithm is also based on an MIS, it inherits the constant approximation ratio from the MOCDS algorithm.

These evaluations provide theoretically useful insights on CDS construction. However, due to those unrealistic assumptions, none of the past research has successfully conducted a CDS performance analysis with practical mobile settings.[2] In contrast, the goal of our simulations is to offer such analysis, and further, to reveal the advantages of our DCDS algorithm in reality.

6.1 Evaluation Setup

We use Random Waypoint [4], a commonly used mobility model, in our simulation. However, the DCDS algorithm is intended to be used in those scenarios with human mobilities, such as a beach, a meeting room, etc. Thus, we set the maximum speed to 5 m/s, the minimum speed to 0.2 m/s, and the pause period is set to 5 seconds. We also fix the bandwidth to 2Mbps.

Instead of using non-stop CBR connections [3, 23], our simulation adopts a random Constant Bit Rate (CBR) traffic model. This CBR model randomly selects a different set of traffic sources and destinations after every few seconds. By assigning different interval values, we are able to simulate different wireless applications that can be deployed in ad hoc networks, from service discovery to multimedia transmissions. We set the packet size to 512 bytes and fix the packet generating rate to 1 pkt/s. We vary the network load by differing the number of CBR connections in every second.

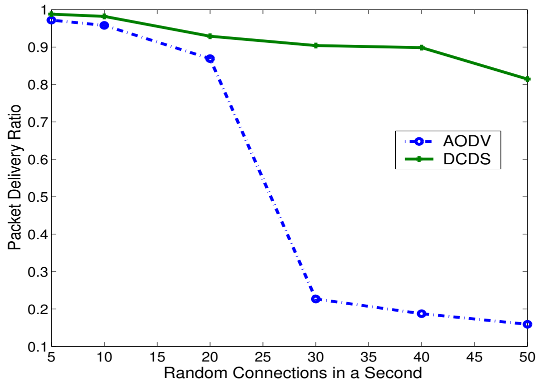
To conduct our evaluations with a large number of nodes in a big terrain area, we use Glomosim [25] as the simulator. However, it has not provided a network model in which the number of nodes increases gradually. Thus, we set the role of every node to dominator at the beginning and deploy DCDS maintenance algorithm to set up a backbone. To prevent this phase from messing up our evaluations, we decide to start the random CBR traffic at second 50, which is the half way point through the simulation.

6.2 Congestion and Interference Effects

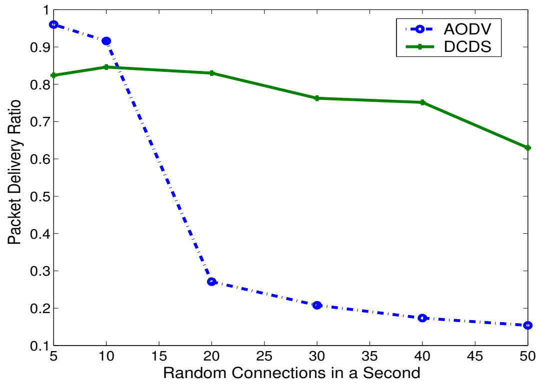
We conduct the first experiment with 100 nodes in a 1500m x 300m area, using a simple service discovery scenario. Service discovery enables users to spontaneously locate surrounding services. For example, a user on a beach may request the locations of nearby volleyball courts, soccer places, wash rooms or restaurants. At present, we are still working on a realistic service discovery pattern. Nevertheless in this paper, we simply set the CBR interval to one second to generate service discovery network traffic. In other words, a CBR source sends out only one packet per second acting as a user sending a service request. We exhibit the Packet Delivery Ratio, as well as detailed congestions and interferences information, in Figure 3 to illustrate the weakness of the flood-and-cache algorithms exemplified by AODV.

Figure 3(a) shows us that the AODV performance degrades dramatically from 87% to 23% as the number of random connections increases from 20 per second to 30 per second with static network settings. AODV, as well as other flood-and-cache algorithms, depends on simple flooding to discover a route from the source to the destination. When a node receives a route discovery packet, it checks whether it is the destination or if it has a refreshed correlated routing information. If so, it sends back a reply, otherwise it has to rebroadcast the packet exactly once. All these broadcast messages interfere with each other and other unicast messages in the airspace. When the service discovery traffic is below 20 requests per second, the carrier-sense mechanism and random backoff procedure can roughly handle it. However, as the traffic increases, they are no longer effective and the flooding storm happens. We can see from Figure 3(c) and 3(e) that AODV suffers from a severe collision and interference problem when there are more than 20 random connections per second in the static network. Figure 3(c) shows the number of signal collisions in the radio layer. This happens when a node receives more than one signal at once, which

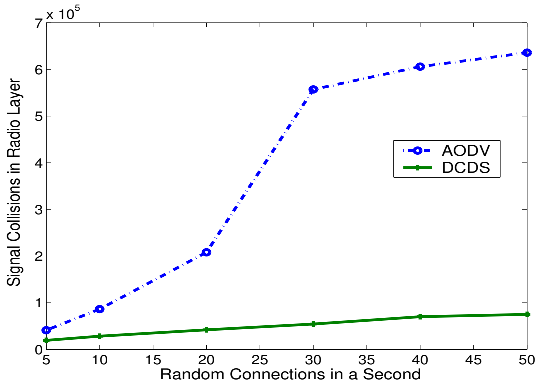
[2] In our simulation of MOCDS, the algorithm virtually never completed the set-up phase.



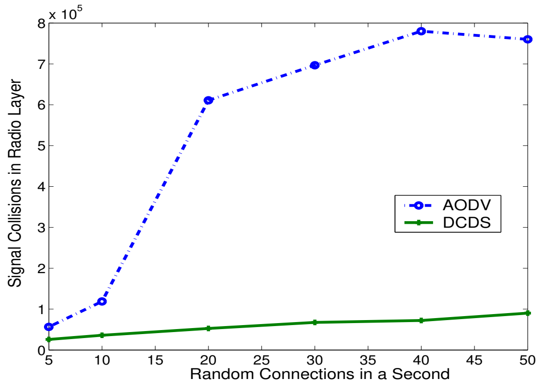
(a) with Static Network Settings



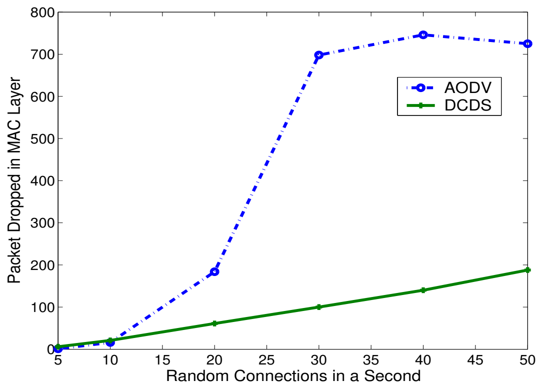
(b) with Mobile Network Settings



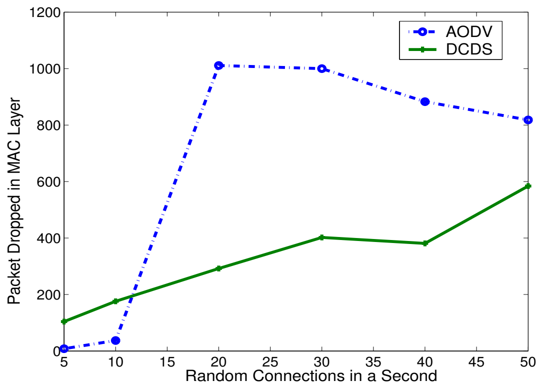
(c) with Static Network Settings



(d) with Mobile Network Settings



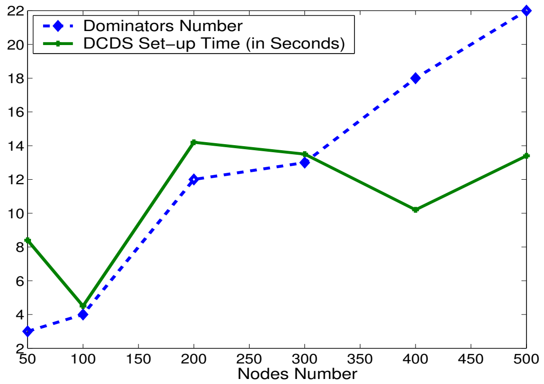
(e) with Static Network Settings



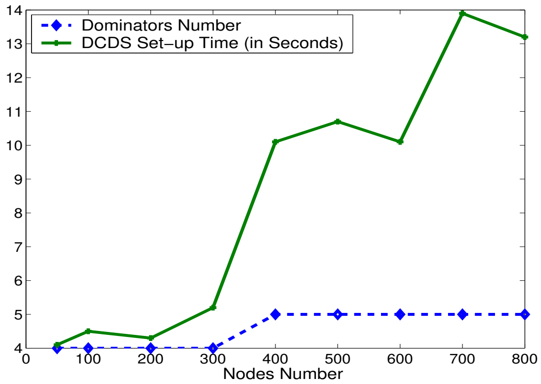
(f) with Mobile Network Settings

Figure 3: Congestion and Interference Effects with a Service Discovery Scenario

is well known as the "Hidden Terminal Problem" [26]. Figure 3(e) shows the number of packets dropped in the MAC layer. This occurs when a unicast packet delivery exceeds its retry limit. As shown in Figure 3(b), this collapse happens even earlier with mobile network settings. The packet delivery ratio of AODV drops from



(a) with Increasing Area and Fixed Node Density



(b) with Fixed Area and Increasing Node Density

Figure 4: Backbone Set-up Time

91.6% to 27.1% when we increase the number of random connections from 10 per second to only 20 per second. Figure 3(d) and Figure 3(f) prove that it is due to same reasons. Moreover, in both static and mobile networks, there are a large amount of packets dropped in the IP layer due to IP queue overflow. This is because of the congested airspace, where a node can hardly determine an idle medium when it attempts to send a packet out. Thus, many packets are stuck in the IP queue resulting in overflow.

On the other hand, Figure 3(a) and Figure 3(b) exhibit that DCDS is much more scalable compared to AODV in both static and mobile networks. This is because DCDS has approximately constant control message overhead regardless of network traffic. Therefore, as shown from Figure 3(c) to Figure 3(f), the number of congestions and interferences increases linearly with the increase of data packets. However, we need to point out that AODV has a better delivery ratio than DCDS at some point as the life time increases. This is because when we fix the network load and extend a CBR's life time, the total number of CBR connections decreases. Thus, AODV gains advantages by reducing the overhead spent on route discoveries. On the other hand, DCDS requires each node to proactively send heartbeat messages regardless of the number of CBR connections. Therefore, it hardly can get any benefit from this reduction. However, we have conducted experiments to prove that the performance of AODV is severely affected by the network congestions when the network is overloaded. Due to space limitation, we are not able to demonstrate these experiments and provide detailed explanations here.

6.3 CDS-related Statistics

The next set of experiments demonstrate the practical DCDS properties in IEEE 802.11 wireless networks. We focus our evaluations on two backbone aspects: set-up time and size.

We regard the backbone set-up time as the interval from the beginning to the point that any two dominators can reach each other. This experiment is conducted only within static wireless networks, where we can easily verify a backbone's correctness. Figure 4 exhibits that the set-up time fluctuates between 4 seconds to 15 seconds as we increase the area by 10 times or the node density by 16 times. Figure 4(a) shows that the growing backbone complexity does not dominate the set-up time. Instead, the set-up time is decided locally by the time a dominator takes to connect to its neighbouring dominators. However, there are many factors that can actually affect the set-up time, such as the intervals for heartbeat messages and timeouts, backoff jitters, node distribution and node density. Considering that we have fixed those intervals and jitters in our implementation, the set-up time largely depends on the latter two causes. We can see from Figure 4(b) that intensifying node density generally extends the set-up time. This is also due to the increasing probability of network congestions and interferences. However, the first jump from 5.2 seconds to 10.1 seconds when the number of nodes increases from 300 to 400 is largely due to the node distribution. In this case, a node with a large ID becomes a dominator. However, it regards itself as a dominatee at the beginning because it receives the DOMINATOR messages from its neighbours with lower IDs. Therefore, it has to wait until the correlated timers are out, which takes about 5 seconds.

To explore the relation between the CDS size and the MIS size, we keep the node density constant but increase the number of nodes and the simulation area by eight times in our last experiment. As shown in Figure 5, the CDS size increases much faster compared to the MIS size: eight-times bigger area adds only 15 more dominators from 3 nodes to 18 nodes; while the number

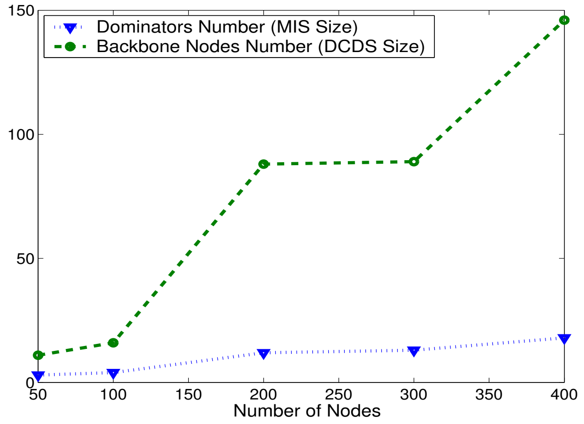


Figure 5: **Flooding-Backbone Routing Scalability**

of backbone nodes increases by 13.3 times from 11 to 146. This is because DCDS asks each dominator to be locally fully connected. Therefore, the number of backbone connections, as well as the CDS size, increases at a faster speed if there are more neighbouring dominators within a three-hop distance.

7. FUTURE WORK

There are three directions for future work. The first attempt is to improve upon the scalability of our routing algorithm. The current DCDS routing is based on a backbone flooding scheme. This intrinsically raises scalability problems if the number of data packets or backbone complexity increases. For example, the DCDS packet delivery ratio drops from 89.9% to 26.1% as we double both the number of nodes and the simulation area from 100 nodes in 1500m x 300m to 200 nodes in 2122m x 425m. This is due to the significant increase of the backbone complexity. After we change the simulation parameters, the numbers of backbone nodes and backbone connections increase from 16 and 10 to 88 and 62 respectively. We plan to investigate the performance of a spanning tree routing scheme (broadcast the backbone through a spanning tree) and an "omniscient" routing scheme (a dominator keeps track of the routes to all the nodes) in the near future.

We are interested in exploring the cache utility for DCDS as well. At this point, a dominator simply drops a data packet when a delivery fails. On the other hand, it can resend a data packet through the new emerging backbone connections if it uses caches, and thus potentially is able to improve its performance. However, we need caution here. Unscrupulously resending a data packet can deteriorate the congestion and interference problem in a mobile wireless network, especially when we deal with the limited 2Mbps bandwidth. At present, we are working on useful heuristics for DCDS to exploit cache functionality in its routing algorithm.

Another intriguing topic we are looking into is making a practical service discovery and delivery model for some real scenarios, such as a beach. We are curious about two issues. One is how the DCDS and other flood-and-cache routing algorithm perform in this model. Another is how we can take over the advantages of both routing schemes in reality.

8. CONCLUSION

This paper presents DCDS, a distributed and localized CDS algorithm, that is deployable for a mobile network based on any current wireless technologies. It is the first CDS algorithm that we are aware of that relaxes three commonly adopted assumptions: reliable broadcast, accurate neighbouring information and a static setup phase. Further, it provides a detailed and effective CDS maintenance scheme to adjust a backbone to any topology changes. In this scheme, each node believes and makes its decisions based upon the local information from its own perspective. We allow a node to make wrong decisions. However, we argue that, through periodic heartbeat messages, DCDS will eventually fix these mistakes. The simulation demonstrates that the proposed algorithm provides a more scalable routing functionality compared to the flood-and-cache algorithm exemplified by AODV, by reducing the probability of network collisions and interferences. DCDS also shows its good properties as the network scale increases, although we need to continue exploring the alternative routing algorithms to improve its performance based on a limited 2Mbps bandwidth.

9. REFERENCES

[1] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, ANSI/IEEE Std 802.11, 1999 Edition*, LAN MAN Standards Committee of the IEEE Computer Society Std., 1999.

[2] C.Perkins and E.Royer, "Ad-hoc on-demand distance vector routing," in *Second IEEE Workshop on Mobile Computing Systems and Applications*, Feb. 1999, pp. 90–100.

[3] S.Das, C.Perkins, and E.Royer, "Performance comparison on two on-demand routing protocols for ad hoc networks," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM'00)*, Tel Aviv, Israel, Mar. 2000, pp. 3–12.

[4] D.Johnson and D.Maltz, "dynamic source routing in ad hoc wireless networks," in *chapter 5, Mobile Computing*. Kluwer Academic Publishers, 1996, pp. 153–181.

[5] J.Broch, D.Johnson, and D.Maltz, "The dynamic source routing protocol for mobile ad hoc wireless networks," Internet-Draft, draft-ietf-manet-dsr-03.txt, 1999, Work in progress.

[6] C.Perkins and P.Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers," in *Proceedings of ACM SIGCOMM'94*, Aug. 1994, pp. 234–244.

[7] J.Wu and H.Li, "On calculating connected dominating set for efficient routing in ad hoc wireless networks," in *Proceedings of the 3rd ACM International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communication*, 1999, pp. 7–14.

[8] P.-J. Wan, K. Alzoubi, and O.Frieder, "Distributed construction of connected dominating set in wireless ad hoc networks," in *IEEE INFOCOM*, June 2002.

[9] Y.Chen and A.Liestman, "Approximating minimum size weakly-connected dominating sets for clustering mobile ad hoc networks," in *The Third ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'02)*, June 2002, pp. 165–172.

[10] K.Alzoubi, P.-J. Wan, and O.Frieder, "Message-optimal connected dominating sets in mobile ad hoc networks," in *The Third ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'02)*, June 2002, pp. 157–164.

[11] P.Chen and A.Liestman, "A zonal algorithm for clustering ad hoc networks," *International Journal of Foundation of Computing Science*, vol. 14, pp. 305–322, Apr. 2003.

[12] M.Gerla and J.Tsai, "Multicluster, mobile, multimedia radio network," *Wireless Networks*, vol. 1, pp. 255–265, 1995.

[13] S.Basagni, "Distributed clustering for ad hoc networks," in *Proceedings of the 1999 International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN'99)*, June 1999, pp. 310–315.

[14] B.Das and V.Bharghavan, "Routing in ad-hoc networks using minimum connected dominating sets," in *Proceedings of the IEEE International Conference on Communication*, June 1997, pp. 376–380.

[15] K.Alzoubi, P.-J. Wan, and O.Frieder, "Distributed heuristics for connected dominating set in wireless ad hoc networks," *IEEE ComSoc/KICS Journal on Communication Networks*, vol. 4(1), pp. 22–29, Mar. 2002.

[16] J.Wu and F.Dai, "On locality of dominating set in ad hoc networks with switch on/off operations," in *Proceedings of the 2002 International Conference on Parallel Architectures, Algorithms, and Networks (I-SPAN'02)*, May 2002, pp. 85–90.

[17] B.Das, R.Sivakumar, and V.Bharghavan, "Routing in ad-hoc networks using a spine," in *Proceedings of the IEEE International Conference on Computers and Communications Networks'97*, Las Vegas, NV., Sept. 1997.

[18] C.Lin and M.Gerla, "Adaptive clustering for mobile wireless networks," *IEEE J. Selected Areas in Communications*, vol. 15, no. 7, pp. 1265–1275, 1997.

[19] R.Sivakumar, B.Das, and V.Bharghavan, "An improved spine-based infrastructure for routing in ad hoc networks," in *Proceedings of the IEEE Symposium on Computers and Communications'98*, Athens, Greece, June 1998.

[20] S.Guha and S.Khuller, "Approximation algorithms for connected dominating sets," *Algorithmica*, vol. 20(4), pp. 374–387, Apr. 1998.

[21] J. Wu and H. Li, "A dominating-set-based routing scheme in ad hoc wireless networks," *Telecommunication Systems*, vol. 18, no. 1–3, pp. 13–36, 2001.

[22] S.Ni, Y.Tseng, Y.Chen, and J.Sheu, "The broadcast storm problem in a mobile ad hoc network," in *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM)*, 1999, pp. 151–162.

[23] J.Broch, D.Maltz, D.Johnson, Y.-C.Hu, and J.Jetcheva, "A performance comparison of multi-hop wireless ad hoc network routing protocols," in *Proceedings of The 4th ACM International Conference on Mobile Computing and Networking (Mobicom '98)*, Dallas, TX, Oct. 1998, pp. 85–97.

[24] T.Haynes, S.Hedetniemi, and P.Slater, *Fundamentals of Domination in graphs*. Marcel Dekker, Inc., 1998.

[25] X. Zeng, R. Bagrodia, and M. Gerla, "Glomosim: A library for parallel simulation of large-scale wireless networks," in *Proceedings of The 12th Workshop on Parallel and Distributed Simulations (PADS'98)*, May 1998, pp. 154–161.

[26] D.Allen. (1993) Hidden terminal problems in wireless lan's. IEEE 802.11 Working Group Papers.