Design and Analysis of a Connected Dominating Set Algorithm for Mobile Ad Hoc Networks

by

Kan Cai

M.Eng., Northeastern University, 2000

B.Eng., Northeastern University, 1998

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

We accept this thesis as conforming
to the required standard

The University of British Columbia

April 2004

Abstract

Wireless technology such as IEEE 802.11b allows a set of devices to communicate with each other in a peer-to-peer manner by dynamically forming mobile ad hoc networks. Routing in such networks is challenging due to node mobility, low power, constrained bandwidth and limited radio range. Most of the previous works are based on strategies that combine flooding and caching to discover routes proactively or on demand. But these algorithms suffer from scalability problems when there exist many spontaneous and short-term connections. This thesis describes the design and implementation of a backbone routing scheme, DCDS, which is inspired by the previous CDS and DSR algorithms. Like other CDS algorithms, it constructs and proactively maintains a backbone across the network; like DSR, it discovers routes on-demand and uses source routing.

However, DCDS makes significant improvements on each of the algorithms on which it is based. It differs from the previous CDS work in that three key assumptions have been removed to make DCDS truly deployable in an IEEE 802.11 network: reliable broadcast, accurate neighbouring information, and a static setup phase. It differs from DSR in that route discovery is restricted to the backbone instead of flooding the entire network and data packets are delivered via multiple paths on the backbone. We have implemented the DCDS algorithm and simulated it using Glomosim. The evaluations clearly show that DCDS achieves significantly better scalability than DSR in a moderately dense network with reasonable mobility settings.

Contents

List of Tables

List of Figures

Acknowledgements

I would like to express my gratitude to my supervisor, Dr. Mike Feeley, for his continuing guidance, support and inspiration, without which I would have given up somewhere in the middle and not been able to complete my thesis this far.

I'm so grateful to have Dr. Norm Hutchinson involved in my project. I would like to express my thanks to him for his valuable comments and encouragement throughout my research, and specially for his bearance with my persistent bothers.

I would also like to thank all the DSG group members, Dr. Charles Krasic, Dima, Geoffrey, James, Joseph, Chamath, Joon, Ritesh and Suprio. They are not only my good friends, but also help me understand many things along the course.

Finally, I'd like to thank my family for their endless support and bearance with the long-term separation.

Kan Cai

The University of British Columbia
April 2004

Chapter 1

Introduction

Wireless networking has hit its heyday. Personal computing is increasingly shifting from desktops to laptops and from Ethernet to wireless local-area networks. PDAs and cell phones are merging to provide computing, web-access, email and instant messaging services along with standard voice telephony. The developed world may soon be at the point where most people carry wireless-networking devices virtually everywhere they go.

One interesting feature of mobile devices based on 802.11 and similar technologies is their ability to form ad-hoc networks without the aid of wired infrastructure. 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 of a wired infrastructure, by providing multiple wireless hops to reach a wired node.

To exploit this potential, an effective protocol is needed to route packets between nodes that are not in direct radio range of each other. Multi-hop routing protocols for this environment face a twin challenge. First, discovering a multi-hop route connecting the source node to the destination from scratch usually requires an expensive, global flooding of the network. Second, ad-hoc networks are very volatile: nodes are mobile and communication is unreliable. As a result, routing information

gathered by a costly broadcast can quickly become out-of-date.

Numerous wireless, ad hoc routing protocols have been proposed and evaluated that address this problem from one of two perspectives. Proactive algorithms, such as DSDV [1] and TBRPF [2], use periodic control messages to maintain up-to-date routing information and are ready to send a packet anywhere at anytime. This approach has the advantage that global discovery broadcasts are avoided. However, there are two main potential disadvantages. First, proactive algorithms impose a fixed message overhead for control messages, even when the network is idle. Second, they can be relatively slow to adjust to topology changes, because they often must wait until a regularly scheduled message exchange to discover that a node has moved or failed.

Reactive algorithms such as DSR [3] and AODV [4], on the other hand, follow a flood-and-cache approach. Nodes discover routing information on demand using global broadcast and then cache this information for subsequent use. They have the advantage that their on-demand nature can make them quick to detect and adapt to topology changes. In addition, they have no periodic control messages and are thus cheaper than proactive approaches when most packets use cached routes or when the network is mostly idle.

The main disadvantage of reactive approaches is that caching is less effective when nodes move and fail, because this behaviour invalidates cached routes. Invalidating a cached route creates two problems. First, reactive algorithms discover bad routes only when using them to deliver packets, typically at the cost of dropping those packets. Second, discovering new routes from scratch is extraordinarily costly, and so the cache must be put to the best use possible. Otherwise, injudicious use of flooding can clog the shared airspace, creating message storms that render the network temporarily useless [5]. Our simulations show that it is precisely this issue that causes considerable problems for the flood-and-cache algorithms exemplified by DSR.

These two problems frame a classic tradeoff for the current-generation of reactive algorithms [6, 7, 8]. Some algorithms such as DSR aggressively cache as much routing information as possible each time they perform costly global communication. Others such as AODV are much more conservative. When mobility and failure invalidate cached routing information, AODV may perform more route discovery broadcasts than DSR and will thus have higher overhead. DSR, on the other hand, may drop more packets, because its caches may be filled with more invalid routes as time goes by.

Others have observed that each class of algorithm performs well for some workloads and mobility patterns, and poorly for others [9, 10]. As a result, a third class, hybrid algorithms, such as ZRP [11] and SHARP [12], attempt to strike useful compromises between pro-activity and reactivity. These algorithms typically use proactive routing within local clusters of nodes and reactive routing among clusters.

This thesis describes the design and evaluation of a new hybrid protocol called Deployable Connected Dominating Set (DCDS). It is inspired by the previous CDS and DSR algorithms. The key idea of our approach is to use a low-level proactive protocol to maintain a communication backbone that organizes the network into clusters and links neighboring clusters to each other. As suggested in the previous CDS algorithms [13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27], our algorithm 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, DCDS selects a subset of nodes, called the Dominating Set (DS), to build up a backbone across the network. On the other hand, DCDS uses a reactive backbone scheme that delivers packets from the source to the destinations via the cluster leaders.

As the name suggests, DCDS improves all the previous CDS algorithms by removing three unrealistic assumptions: reliable broadcast, accurate neighbouring information, and a static setup phase. Therefore, DCDS is the first CDS algorithm

we are aware of that can be truly deployed in an IEEE 802.11 network.

The reactive part of our algorithm is very similar to DSR. However, the routing caches are stored only on cluster leaders and routes are confined to the backbone. As a result, we are able to cache as aggressively as DSR, while efficiently maintaining cache consistency. We achieve this benefit by limiting the total number of caches and the number of different routes that can be invalidated by movement or failure of a single node. Another benefit of our approach is that route discovery is cheaper than in other reactive algorithms, because we flood only the backbone, not the entire network.

Another improvement on DSR is that DCDS uses a multi-path routing scheme. The data packets originated from a source node can be sent to different dominators in its neighbourhood. Therefore, these packets can be delivered via different routes. Also, the route provided in the packet header is incomplete, which lists only a sequence of all the intermediate dominators. This allows each dominator to choose a dominatee on its own to forward a message to the next dominator specified in the header. Since there are multiple dominatee candidates that connect two neighbouring dominators, DCDS has the potential to use multiple paths for local recovery in case a message delivery fails.

In addition to showing the backbone scalability while increasing the network scale and node density, we evaluated our protocol and compared it to DSR using simulation. These experiments are conducted to evaluate performance by varying node density, mobility, network load and traffic patterns respectively. Our results show that DCDS can provide significantly better performance and generate lower overhead by eliminating network-wide broadcast and aggregating routing information, especially when dealing with short-term CBR connections.

The rest of the thesis is organized as follows. We first briefly discuss DSR and the previous CDS algorithms as well as their problems in Section 2. Then, we detail our DCDS algorithm and the backbone routing scheme in Section 3. In Section 4,

we present the detailed simulation results of DCDS against DSR and also provide a thorough analysis. Finally a conclusion and some ideas for future work are offered in Section 5

Chapter 2

Related Work

2.1 IEEE 802.11 DCF

There are many MAC/PHY protocols that have been proposed to support wireless communication in mobile networks, among which IEEE 802.11 [28] is the most widely accepted. However, at this point, communication among wireless devices using 802.11 is typically based on a wired infrastructure. That is, the access points organize nearby wireless devices and handle all of their communication.

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.

On the other hand, IEEE 802.11 also offers a tantalizing but largely unutilized alternative – *Distributed Coordination Function* (DCF), which allows nodes to self organize to form ad-hoc networks without the aid of any wired infrastructure. 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.

However, avoiding signal interference and message collision is inherently difficult for such ad hoc networks. The fundamental challenge involves management of the shared airspace around each wireless device without a central arbitrator. To address this problem, 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 interference, collisions, or time-varying channel properties [28].

2.2 Flood-and-Cache Routing Algorithms

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 routes 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. [9] demonstrate that reactive algorithms generally outperform proactive ones in terms of delivery ratio and routing overhead for long-lived end-to-end connections. The following section briefly discusses the DSR algorithm, one of the routing schemes on which DCDS is based.

2.3 DSR

The dynamic source routing algorithm (DSR) [3] enables a source node to set up and maintain a multi-hop route to a destination. Each node has a routing table which maintains a set of destination nodes and the complete paths to reach them. It uses an on-demand route discovery and maintenance process using the RREQ, RREP and RERR control messages.

When a mobile node tries to communicate with an unknown destination, it broadcasts a route request message (RREQ) to discover such a route. Each node receiving this packet has to rebroadcast the request exactly once unless it is the destination node or it knows a route to reach the destination. Such nodes report the route information to the source by sending back a route reply message (RREP).

DSR requires each forwarding node to keep track of its next-hop connection

along the route, using any available link or network layer mechanism. In IEEE 802.11 networks, in particular, DSR may utilize the explicit link layer acknowledgements to determine the link availability. If a packet delivery fails, the upstream node regards the link as broken and initiates a route error message (RERR). A RERR packet is relayed by reversing the path indicated in the packet header until the src node receives it. Then, the source node may restart the discovery process if there is no alternative route in its cache.

Compared to other relative algorithms such as AODV, DSR has several major differences and optimizations. First, a source node in DSR uses the source routing scheme to deliver packets. If a forwarder along the path finds a broken link, it attempts to salvage the packet using its own routing information, in addition to unicasting a RERR message back to the source.

Second, DSR takes a very aggressive route caching approach. The source routing scheme not only helps the source node learn the routing information to reach every other node along the path, but also enables an intermediate node to complete its own routing table when forwarding a packet. Further, route discovery in DSR is able to find multiple paths to the target node since the destination node replies to all the requests even if they are actually duplicates from different paths. Finally, DSR takes advantage of the promiscuous function of the network interface. This optimization enables nodes to learn potentially useful routing information by overhearing messages from its one-hop neighbours.

Last, DSR deploys *gratuitous RREP* and *gratuitous RERR* messages to improve its performance. The *gratuitous RREP* message is used to shorten an existing route when a node overhears a message not addressed to itself. If it finds that the source route header includes its address and it has not yet received this message, it removes those intermediate nodes from the route and sends a gratuitous reply message to the source reporting this shorter path. A DSR source node also uses the *gratuitous RERR* message to propagate stale link information to the rest of network.

If a source needs to initiate a RREQ message, it piggybacks the latest broken link that it knows into the request. Therefore, each node receiving this route request updates its routing table, and thus will not generate route replies containing the broken link.

Recent work [8, 7] shows that aggressive caching seems to be a good design choice with certain mobility and traffic patterns by alleviating network congestion. However, it has its own problem, i.e., *cache staleness*. In a dynamic wireless network, any broken link invalidates an entire path. Sending data packets through a outdated path may not only cause packets to be dropped, but also potentially pollute the route caches in other nodes. A number of previous works [29, 30] have shown that the stale cache entries can adversely affect DSR performance. They also proposed some solutions to fix this problem, including packet salvage, gratuitous route repair, wider error notification, time-based or epoch-based route expiry, etc., and some of them have been adopted by the current DSR implementation. However, this thesis shows that aggressive caching with indefinite lifetime is still problematic if there exist many short-lived CBR connections since DSR is not able to detect a broken link promptly with such a traffic pattern.

2.4 Previous CDS Protocols

The previous works [9, 6] have clearly demonstrated that DSR can provide a satisfactory packet delivery ratio for long-term end-to-end CBR connections. However, it is still questionable to deploy such reactive algorithms in scenarios where many spontaneous communications exist. This random and short-lived network traffic can generate heavy overhead and easily trigger a flooding storm [5] due to the broadcasted route discovery messages.

The connected dominating set (CDS) protocol is dedicated to reducing the overhead and addressing this scalability problem. Its distinguishable characteristic is to select only a subset of routing nodes which deal with packet delivery on behalf

of the others. These nodes should connect to each other, organizing a backbone that can reach the remaining nodes in the network. By doing so, a CDS algorithm only needs to retain the aggregated backbone's connectivity using local-scale broadcast within a one-hop distance, instead of maintaining individual routes for each source and destination pair using flooding.

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 [31]. Previous CDS research has thus focused on heuristics for 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*.

An intuitive way to obtain a CDS is to first identify a dominating set, and then grow it into a CDS by adding connector vertices. Gerla et al. [18, 24] present distributed algorithms that follow this approach by using the lowest node ID or the highest node degree to select nodes into the dominating set. These results are generalized by Basagni [19] to use an abstract weight to determine membership. Das et al. [20, 23, 25] decentralize Guha and Khuller's algorithm [26] and come up with a series of algorithms for setting up a network backbone. Wan et al. [14] 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) [15, 17], where two dominators can be separated by a two-hop dis-

tance. Based on Guha and Khuller's centralized spanning tree algorithm [26], 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 [17], and to control the growth of the spanning tree.

Wu and Li [13] 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. [22, 27] 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 [14]. 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 [14, 16, 21] 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 [16] 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.

2.5 Problems in the Previous CDS Algorithms

The MCDS problem has received much attention and a number of algorithms [13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27] have been proposed to achieve a small CDS subset with good approximation quality. These algorithms provide useful insights on how to construct and maintain a CDS backbone in mobile ad hoc networks. However, they also share the same unrealistic assumptions that prevent implementation with the existing wireless technologies, IEEE 802.11 in particular.

The first assumption is that each node should have accurate topology in-

formation gathered from broadcast messages. Generally speaking, these algorithms either designate some centralized nodes that identify all the topology changes in the network or require each individual node to keep track of all its neighbours. Such "omniscient" topology information enables nodes to build a backbone and adjust it in a dynamic environment.

Second, a reliable MAC-layer broadcast scheme is assumed to effectively send control messages. This is because a node has to synchronize with all the other correlated nodes, at least its neighbours, before it accommodates any topology changes. For example, a node can declare itself as a cluster leader only when it receives the implicit or explicit consent of all its neighbours. The process may fail if any such message delivery fails. Worse, it could even end up in a situation with all the nodes trapped into deadlocks.

Third, the previous CDS algorithms assume that there is a distinction between the backbone construction phase and maintenance phase. The construction phase requires a global static snapshot, where no topology change is allowed, to establish the initial backbone. Afterwards, only a local static neighbourhood is needed to maintain the graph connectivity.

These assumptions facilitate theoretical analysis on these CDS algorithms in terms of approximation ratio, time complexity and message complexity; but also because of them, it is hard to deploy a backbone routing scheme for wireless applications. Obtaining accurate and complete topology knowledge is fundamentally complicated in a mobile ad hoc network without support from any outside infrastructure. Depending on an unreliable broadcast function to propagate and aggregate such information makes it even worse. IEEE 802.11 cannot provide reliable broadcast, and thus it is also difficult to synchronize with other nodes, a node's neighbours for example, as required by the second assumption. Furthermore, the normal case for an MANET is not to start from scratch with a large number of static nodes. Instead, the typical case is to add new nodes into an existing network, where

topology changes and node exceptions are inevitable. Hence, it is inappropriate to assume a static global-scale or local-scale snapshot for backbone construction and maintenance.

Chapter 3

DCDS Algorithm

Similar to other hybrid algorithms, DCDS is partly proactive and partly reactive. Its proactive component acts to maintain a single spanning graph for the network. Routing is performed reactively in a manner similar to DSR, but where routes are confined to follow the spanning graph.

The graph is constructed using a variant of the Message-Optimal Connected Dominating Set Algorithm [16], modified to work incrementally and to be resilient to communication failures. The algorithm selects certain nodes to act as dominators; all other nodes are called dominatees. The dominators cover the network so that every dominatee is within radio range of a dominator and no two dominators are in range of each other. The graph links dominators together using two- and three-hop links.

The role of the graph is to confine routing to a small subset of the network. This approach has two key advantages. First, it lowers the cost of route discovery. DCDS discovers new routes by flooding the spanning graph, not the entire network. Second, it improves the mobility-failure resilience of route caching. DCDS reduces the number of cached routes invalidated when a node moves or fails by using fewer caches and fewer routing nodes, compared to DSR and similar algorithms.

In DCDS, caching and routing are handled exclusively by dominators, a rel-

atively small subset of the network. When a non-dominator moves or fails, the only routes that might break are routes to that node. In DSR, on the other hand, the fact that every node is a routing node means that a similar failure can invalidate many more routes: routes to the failed node and routes that pass through it. Similarly, DSR caches routes on every node and thus it potentially stores many more copies of each route somewhere in the network.

Our approach does present a fairness problem. Much of the network load is focused on nodes in the backbone. If the network is comprised of peers, a node pays a high price if it is selected to be part of the backbone. This is a well known problem with connected dominating set algorithms like ours. But addressing this issue is left for future work. Note, however, that our approach does not cause the network to be underutilized. Even though we confine routes to the spanning graph, this graph will cover virtually the entire airspace of the network and will thus efficiently utilize available bandwidth.

This chapter describes the design of the DCDS algorithm in two parts corresponding to the two abstract layers of which it is composed. We begin by describing the upper layer that reactively routes packets among arbitrary nodes using the spanning graph. We then describe the lower layer that proactively maintains the spanning graph and supports communication between a dominator and its neighboring dominators or in-radio-range non-dominators.

3.1 Reactive Backbone Routing

The basic operation of the routing protocol is to deliver payload packets from source nodes to their targets. To send a message, a source node assembles a packet consisting of target-node ID and payload, and sends it to an in-radio-range dominator; if multiple dominators are in range, it chooses one arbitrarily. The receiving dominator checks its cache for the target and initiates route discovery if necessary, buffering the packet in the meantime. Once the dominator has a route to the target, it adds

this sequence of dominator-node IDs to the packet and hands the packet to the backbone layer for delivery to the first dominator on the path. At each dominator, the lower layer upcalls the routing layer and then delivers the packet to the next dominator on the path. The lower-layer on the last dominator delivers the packet to the target node.

We now cover the key features of the routing layer — caching, discovery and error handling — in more detail.

3.1.1 Route Caching

The nature of the underlying backbone suggests the two-part caching scheme we follow. Routing consists of two steps: locating a dominator in range of the target node and planning a backbone route to that dominator. These two steps are supported by distinct caches stored on each dominator. The *dominatee routing table*, DRT, caches dominatee-dominator pairings; the *backbone routing table*, BRT, caches backbone topology information. Dominators maintain their DRT and BRT reactively by observing the content of messages they receive, either as the target or as a routing node.

The DRT is indexed by dominatee ID and lists one dominator and a timestamp for each dominatee in the table. It is updated by route reply messages, which typically include a list of multiple dominatee-dominator pairings and corresponding timestamps, as described in the next section. When a dominator receives such a message, either because it requested the discovery or because it is forwarding the reply to another dominator, it adds the pairing information in the message to its DRT. Timestamps ensure that newer pairings replace older pairings. For simplicity, each dominatee in the table is listed with a unique dominator, even though it may be in range of several simultaneously, and entries are never deleted, though timestamps could be used to prune old entries if the table grows too large.

The BRT stores a set of backbone paths. It is updated by every packet

a dominator receives and thus normally captures the current backbone topology quite accurately. When a dominator receives a packet, it merges the *path-traveled* information the packet contains into its BRT. The routing layer adds this information to packets as it routes them from dominator to dominator along the backbone; reply messages also include the path traveled from requesting to replying node. When a dominator receives an error packet that invalidates a particular backbone link, it deletes the link from paths in its BRT.

3.1.2 Route Discovery

As mentioned above, routing begins with a source sending a data packet to a dominator. This dominator checks its DRT for a dominator in range of the target node. If found, it then checks its BRT for a route to this target dominator. If either of these two cache lookups fails, the source dominator buffers the packet and initiates a route discovery for the target node. If it does not receive a corresponding route reply within a timeout period, it drops the packet.

The source dominator multicasts a ROUTE DISCOVERY message to the entire backbone. This multicast delivery is implemented by the lower backbone layer, which uses unicast messages to flood the packet to every dominator reachable from the source.

When a dominator receives a discovery message, it checks a *dominatee ownership table*, DOT, maintained by the lower level to see whether the target node is in its radio range. If not, it then checks its DRT and BRT to see whether it knows which dominator the target node belongs to. Finally, if it knows nothing about the target, it adds itself to the path-traveled information in the packet header and instructs the lower layer to continue flooding the request to nearby dominators.

When a dominator locates the target, it sends a ROUTE REPLY packet back to the requesting dominator. This packet is routed by reversing the path-traveled information encoded in the request, and has the complete backbone information

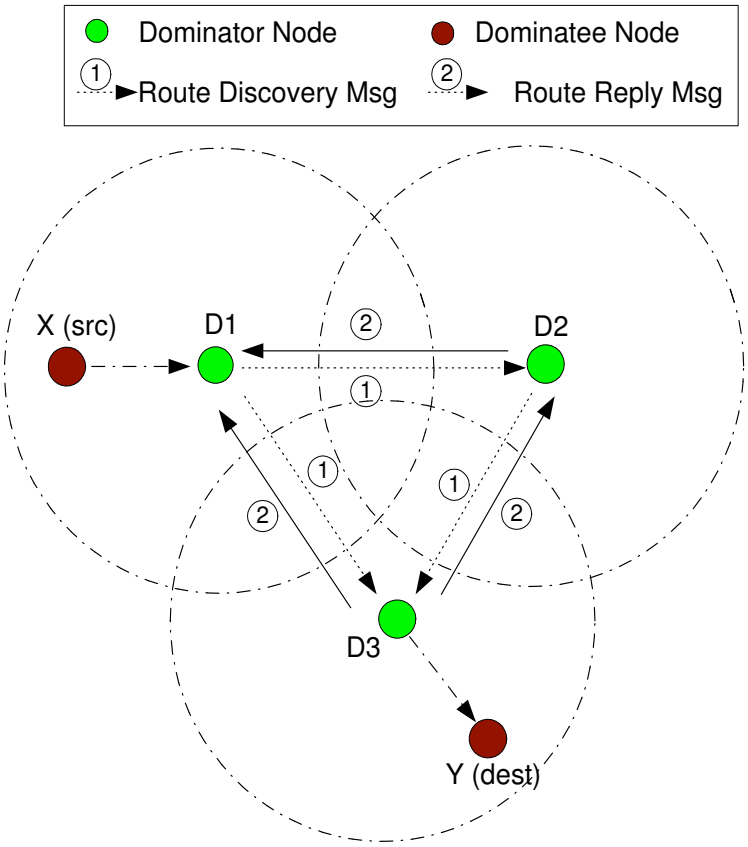


Figure 3.1: Example of Route Discovery.

from the requesting dominator to the target dominator. The reply also includes a timestamp, extracted from its DOT or DRT, that indicates when the target dominator first discovered the target dominatee. As explained in the previous section, the requesting node compares this timestamp to information already in its DRT to ensure that it stores the most recent dominator for the dominatee.

Figure 3.1 shows a simple example of the route discovery process initiated when node X attempts to send a packet to node Y. The network consists of three dominators: $D1$, $D2$ and $D3$; circles around the dominators indicate their radio range. Directed edges indicate message sends: requests are labelled "1" and replies

"2". If we assume that the BRT and DRT of every node are initially empty, then when $D1$ receives the message from X, it multicasts a ROUTE DISCOVERY message looking for the node Y. In the first step of the multicast, the backbone sends the request to nodes $D2$ and $D3$. Both check their DOTs for Y. $D3$ finds it and thus sends a ROUTE REPLY packet back to $D1$. $D2$ does not not find Y and so it continues the multicast by sending the request to $D3$. $D3$ replies to $D1$ again, but this time along the path that includes $D2$. As a result, $D1$ and $D2$ update their DRTs to record that Y is in range of $D1$. $D1$ and $D3$ update their BRTs to reflect the complete backbone topology, while $D2$ is only missing the path $D1 - D3$.

Finally, as an optimization, route reply packets are padded to piggyback dominator information for multiple dominatees. The target dominator uses an open-ended vector of timestamps in the route request along with its DOT timestamps to determine which of its dominatees are unlikely to be in the requester's DRT. It appends the IDs and timestamps of these dominatees to the reply packet, if there is room. The vector of timestamps is encoded by the requesting node to summarize the current state of its DRT with respect to other dominators. We treat the timestamp vector as an optional heuristic. If the target does not find a matching entry in the vector, it pads the reply with as many DOT entries as will fit, starting with the newest. Other dominator nodes on the reply path repeat this process as long as room remains in the reply packet.

3.1.3 Routing Errors

The routing layer relies on the backbone to determine which nodes are dominators, which links exist between dominators, and which dominatees are in range of each dominator. The backbone only updates this information periodically based on heartbeat messages, as will be described in Section 3.2, and is thus sometimes out of date. Some errors are thus discovered only when the routing layer is attempting to send a message. In two cases the routing layer performs corrective action on-the-fly

in an attempt to salvage an otherwise errant message send.

The first case occurs when a packet is routed over a backbone link between two dominators that are no longer connected. When this occurs, the backbone delivers a ROUTE NACK message to the dominator that attempted to use the faulty link. This dominator deletes the bad link from its BRT, forwards the ROUTE NACK backward to the source dominator, and attempts to salvage the packet. Dominators buffer a fixed number of recently sent packets for this purpose. If the packet is buffered, the dominator checks its BRT for an alternate backbone route to the target node. If it finds one, it updates the route information in the packet header and resends the packet, decrementing a SALVAGE header field to limit the number of times the packet can be redirected in this way.

The second case occurs when a source dominator has outdated DRT information for a target node. To deal with this case, every dominator on the packet's route checks its DRT to determine whether its entry for the target is more recent than that used to route the packet; packets include their target node's DRT timestamp for this purpose. If a dominator does have more recent information, and it has a BRT path to the target's new dominator, it updates the packet's route information before forwarding it.

3.2 Proactive Backbone Maintenance

We now turn to the lower-layer of the protocol which proactively maintains the spanning graph used to route messages. The basic idea is to group nodes into clusters around a *cluster-head* (i.e., a dominator) chosen according to some globally consistent formula; in our case the node with the lowest ID in its one-hop neighbourhood is a dominator. A dominator uses its dominatees to connect its cluster to the nearby clusters via either two- or three-hop paths. Periodic heartbeat messages are used to maintain the clusters, their cluster heads, and inter-cluster links.

Any node can act as either a dominator or dominatee, with some dominatees

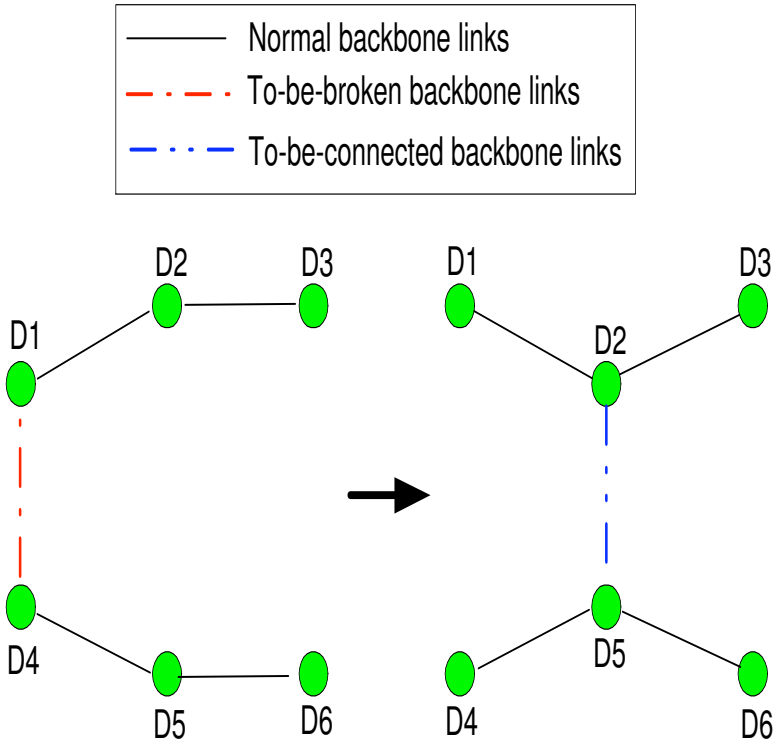


Figure 3.2: The Fake Partition Problem

acting as *connectors* that link dominators to each other. Dominators store a list of in-radio-range dominatees in the *dominatee ownership table* (DOT). As described in the previous section, each DOT entry is timestamped when added. Dominators also store a *connectivity list* containing paths to other dominators that are two and three hops away. Similarly, dominatees store a list of in-range dominators, timestamped each time a message is received from that dominator, and a connectivity list containing paths to dominators that are at most two hops away. Finally, outdated information is purged from the various local lists unless it is periodically refreshed by the messages described below.

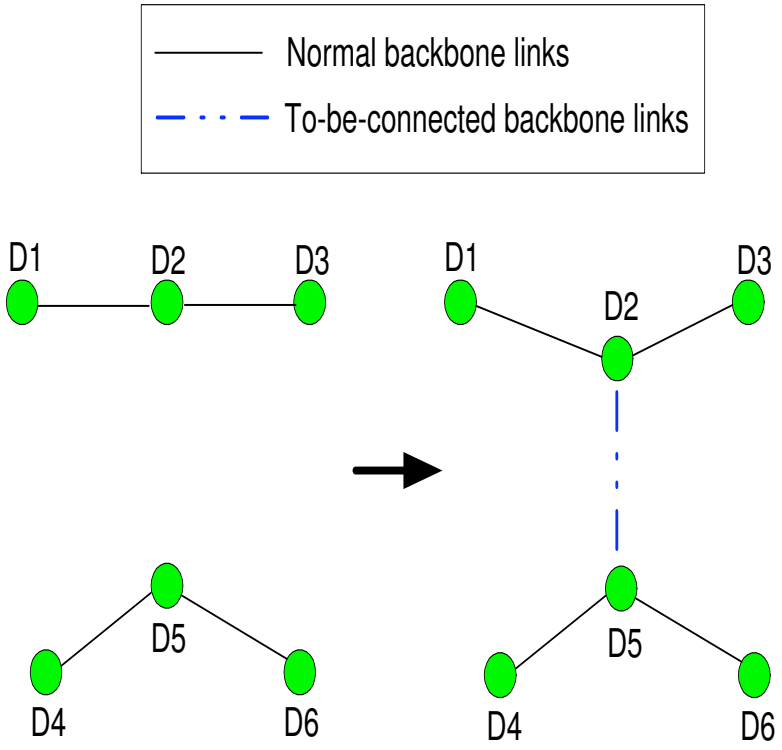


Figure 3.3: The Failed Merge Problem

3.2.1 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 use localized CDS maintenance schemes [16, 27], but they do not provide ways to detect topology changes or identify moving nodes. We also attempted to define 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 mobility settings, local reactive repair 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 3.2 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, $D2$ and $D5$ are moving towards 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 3.3, 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 a 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 structure so that they can piggyback the current backbone connectivity information.

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* and *Dominatees list* , and these heartbeat messages are responsible for refreshing 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 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 decisions 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 might not be delivered successfully, the successive heartbeats can eventually fix all the consequences.

3.2.2 Constructing the Graph

The graph is constructed inductively. By default every node is a dominator until it discovers another dominator in its radio range that has a lower ID.

Each dominator periodically broadcasts a DOMINATOR message that is received by every node in its radio range. This message includes the dominator's ID and a list of its one-hop neighbors in its connectivity list, which is initially empty. When a dominator node receives a DOMINATOR message with an ID lower than its own, it changes its state to dominatee. When a dominatee receives a DOMINATOR message, it records the dominator in its local list and updates the dominator's timestamp.

Each dominatee periodically broadcasts a DOMINATEE message that is received by every node in its radio range. This message includes the dominatee's ID and its lists of one- and two-hop dominators, which are initially empty. These messages are sent much less frequently than DOMINATOR messages and constitute the major message overhead of this proactive approach.

When a dominator receives a DOMINATEE message, it does several things. First, it adds the dominatee to its DOT, if it is not there; it timestamps new DOT entries with the current time. Second, it adds the dominators listed in the message to its connectivity list. For the two-hop-away dominators, the dominatee is the two-hop connector. For the three-hop-away dominators, the dominatee is the first hop of a three-hop connection. The second hop is selected at the dominatee from its local connectivity list.

When a dominatee receives a DOMINATEE message, it adds only the one-hop dominators in the message to its connectivity list; the initiating dominatee is the connector for these links.

If a dominatee node fails to receive any DOMINATOR messages for a sufficient interval, it may need to initiate an election to select a new dominator. The failure to receive a DOMINATOR message, however, is not a sufficiently strong indicator that there are no dominators in range. The reason for this is that 802.11 gives priority to unicast messages and thus broadcast messages such as DOMINATOR can be drowned out during periods of sustained congestion.

Therefore, before initiating a dominator election, the dominatee sends unicast ping messages to check whether any of the dominators in its local list are reachable. Those that fail to respond to the ping are deleted from the list. Only if none of them responds does the dominator change its state to dominator and broadcast a DOMINATOR message.

3.2.3 An example

Figure 3.4 shows an example of a simple backbone that consists of three nodes that will eventually become dominators — $D1$, $D2$ and $D3$ — and three that become dominatees — $C1$, $C2$ and $C3$. The circles around the dominators represent their radio range. The boxes summarize the information stored at each dominator once the protocol has reached steady state.

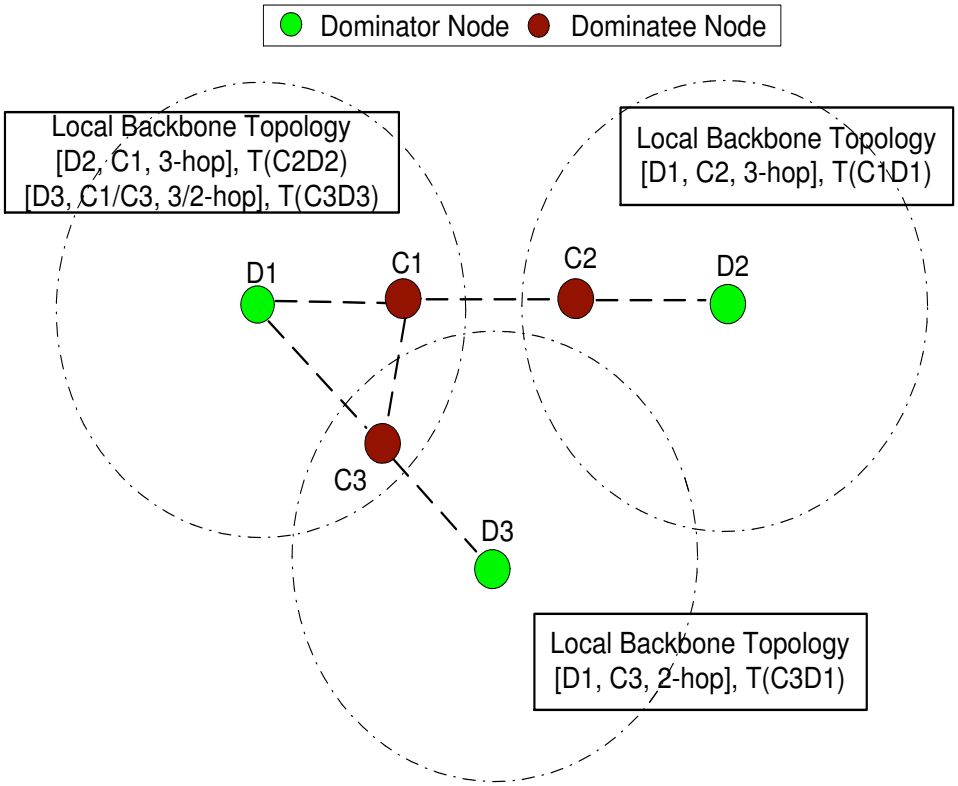


Figure 3.4: Example of a Simple Backbone.

In the first round of messages $C1$, $C2$ and $C3$ transition to dominatee when they receive DOMINATOR messages from one of the other nodes. $C1$'s dominator list contains $D1$, $C2$'s contains $D2$, and $C3$'s contains $D1$ and $D3$. All other lists are empty until $C1$, $C2$ and $C3$ send their DOMINATEE messages. When they do, the dominators update their DOTs and $D1$ and $D3$ add each other to their connectivity lists, with $C3$ as the two-hop connector. In addition, $C1$ adds $D2$ to its connectivity list and $C2$ adds $D1$, each listing the other as the connector. Finally, in the next round of DOMINATEE messages, $C1$ includes $D2$ as a two-hop dominator and $C2$ includes $D1$. These messages allow $D1$ and $D2$ to establish a three-hop backbone link through $C1$ and $C2$.

3.2.4 Multi-path Backbone Routing

While the routing layer treats backbone links between dominators as single paths when constructing routes, in reality each link is a multi-path connection involving one or two connector nodes. Where there are multiple low-level connections that can instantiate a upper-level path, the low-level routing protocol is afforded flexibility when dealing with link failures.

The basic backbone routing between two dominators works as follows. An upstream dominator checks its connection list for connections to the next dominator. If two-hop connections exist, it chooses the one with the most recent timestamp, otherwise it chooses the most recent three-hop link. In either case, the result is that a connector node in range of the dominator is chosen. The dominator then sends a unicast message containing the payload to the connector. A two-hop connector directly sends the packet to the target dominator, while a three-hop connector repeats the process to select a second connector.

If a dominator or connector is unable to send the packet, it receives a MAC-protocol-level error; connectors forward the error to the upstream dominator. When nodes receive such an error, they immediately delete the errant connection from their connectivity lists. The upstream dominator, which buffers recently sent packets, selects another connection and tries again. Only when the retry limit is reached or all available connections have been deleted is the error reported to the upper routing-level protocol. Recall that this error triggers an attempt to salvage the packet at that level if alternate backbone routes exist to the packet's ultimate destination.

Chapter 4

Evaluation

Previous CDS work [13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27] evaluates a CDS backbone by its approximation ratio, and a good CDS algorithm should have its approximation ratio bounded by a constant.

However, these evaluations have failed to conduct CDS performance analysis with practical mobility settings due to some unrealistic assumptions they have made, as discussed in Section 2.4. In contrast, the goal of our simulations is to offer such analysis, and further, to reveal the advantages and disadvantages of our DCDS algorithm in an IEEE 802.11 ad hoc network with a reasonable mobility and traffic model.

4.1 Simulation Setup

We conduct the simulation using Glomosim [32], a scalable simulator with accurate physical layer and radio propagation models. In our simulations, the bandwidth is set to 2 Mbps with 2.4 GHz radio frequency and the transmission range is set to $250m$. Table 4.1 lists the main constants used in our DCDS

Our evaluation compares DCDS to DSR. To ensure our comparison was as faithful as possible to previously reported DSR results, we imported the $ns2$ DSR code from the Monarch project [33] and modified it to work with Glomosim. We

Table 4.1: DCDS Protocol Settings

DOMINATOR Heartbeat Interval	0.5 Second
DOMINATEE Heartbeat Interval	5 Seconds
Dominator Timeout	5 Seconds
Dominatee Timeout	30 Seconds
Retry limit for local recovery	4
Retry limit for salvage	2
Time to hold packet awaiting routes	40 Seconds
Timeout for buffering a packet after forwarding	5 Seconds
Timeout for awaiting for a dominator AYA ping response	1 Second

verified that this implementation closely matches the $ns2$ version when physical values are set as suggested by [34]. implementation.

The evaluations are conducted with a total of 200 nodes that are randomly distributed in an area of $1500m$ x $750m$. We choose an area that is 3 times as large as that used in the previous work [9, 6] in order to avoid a chain-like backbone, which would tend to favour a backbone approach such as ours. By using static mobility settings and varying the seed value provided by Glomosim, we are able to accurately evaluate the backbone topology with different node distributions in this network. With 10 trials, there are approximately 14 dominators in the network, and each dominator is on average connected to 7 neighbouring dominators.

We use Random Waypoint [3] to model mobility as is common. Using this model, each node randomly chooses a destination and moves towards it with a velocity chosen randomly from $[V_{min}, V_{max}]$. Each simulation lasts 910 seconds. The minimum speed is set to 1.0 m/s and the pause period is set to 60 seconds. Since the evaluation is mainly intended to investigate performance in scenarios with human mobility, we set the maximum speed to 5 m/s in most of the simulations. However, in section 4.6, we give DCDS performance for a variety of faster mobility settings.

We adopt a multi-destination CBR traffic pattern similar to that used in the SHARP paper [12]. This pattern randomly selects a set of destinations to act as communication hot spots. The number of hot spots is a parameter that we vary. Source nodes are chosen randomly from the network and destination nodes are chosen randomly from the list of hot spots. The duration of each CBR connection is a parameter that we vary. When one ends, another is chosen to take its place. We are thus able to simulate a variety of wireless environments by varying the connection duration. The size of each CBR packet is 256 bytes and packets are generated at the fixed rate of one packet per second. We vary network load by changing the number of concurrent CBR connections.

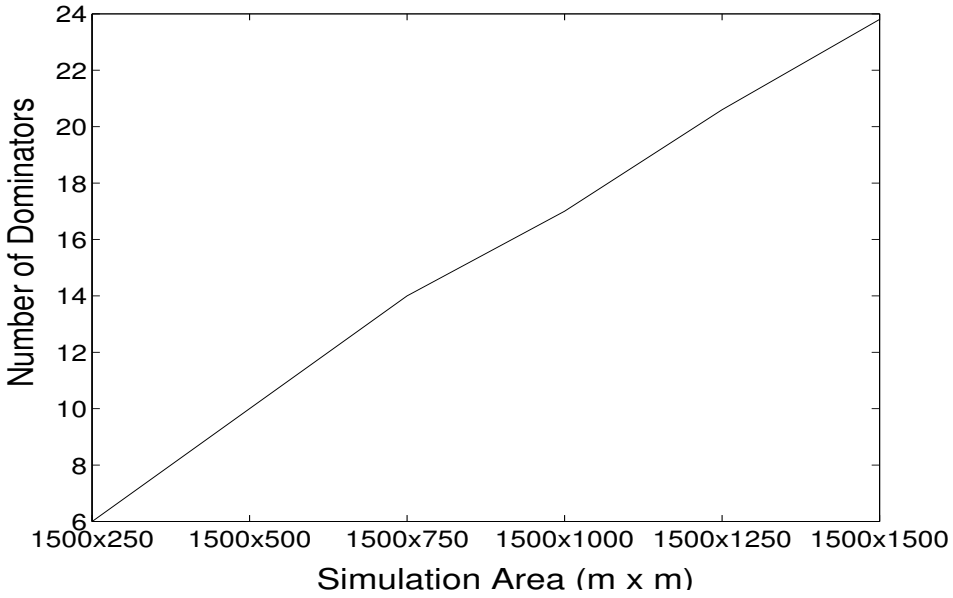
To eliminate startup and shutdown effects, each CBR source starts sending data 50 seconds into the simulation, and stops at 900 seconds, 10 seconds before the end of the simulation. The start time of every CBR source is perturbed randomly by at most one second to reduce the probability that their synchronization causes unnatural congestion.

Finally, the main metrics we use in our comparison are packet delivery ratio and message overhead. Delivery ratio, a common metric, gives the fraction of packets successfully delivered compared to the number sent.

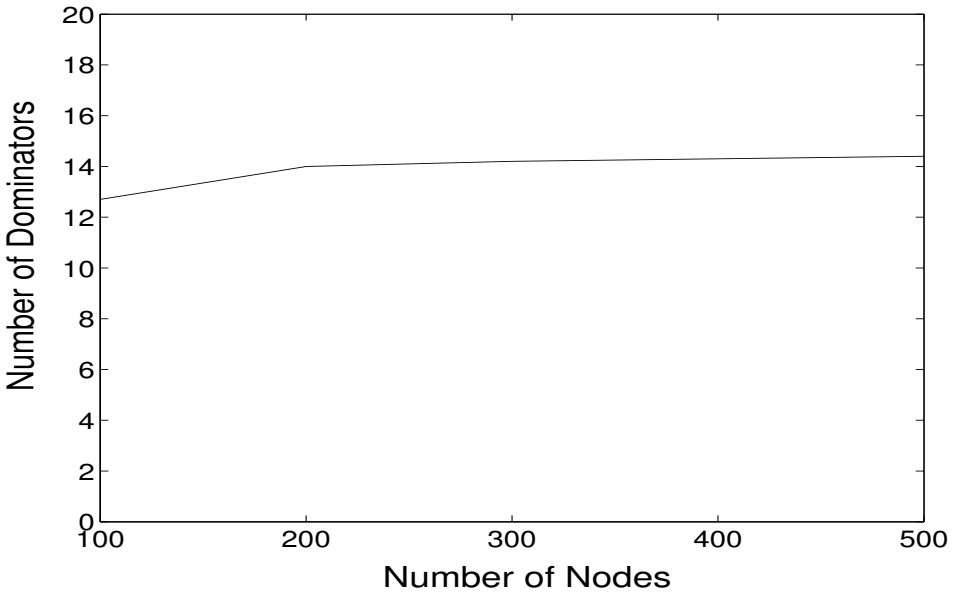
4.2 Backbone Scalability

Before we show DCDS routing performance in mobile ad hoc networks, we investigate the backbone scalability in two aspects. The first aspect we measure is the growth of the number of dominators as we increase the network size or the node density. The second aspect we measure is the growth of the dominator degree, i.e., the average number of neighbouring dominators that each dominator has. The smaller the two aspects are, the more advantages the DCDS can achieve.

It's worth noting that since the DCDS algorithm is based on the unit disk graph, it has a constant approximation ratio. This has been theoretically proven by

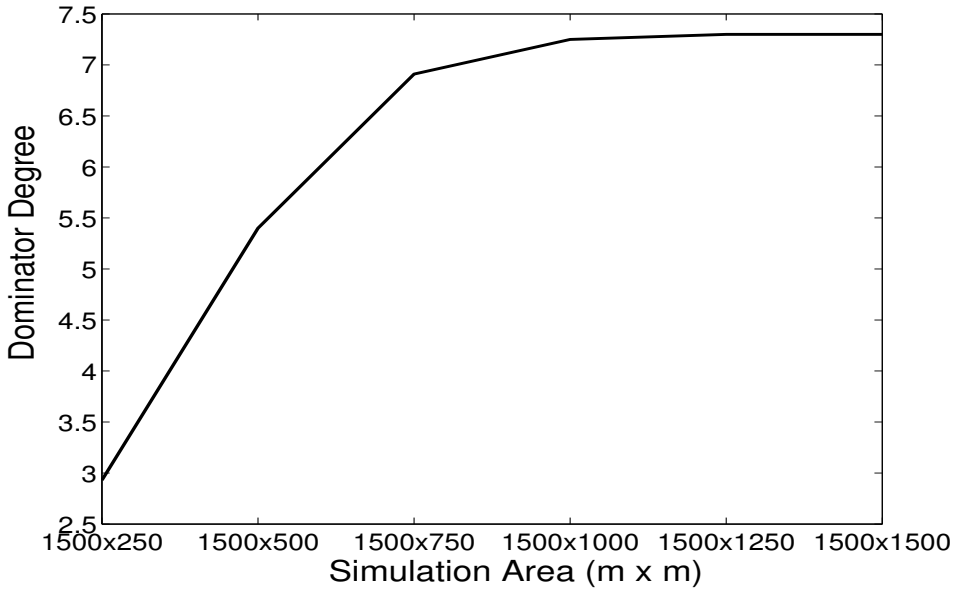


(a) With 200 Nodes and Increasing Network Area

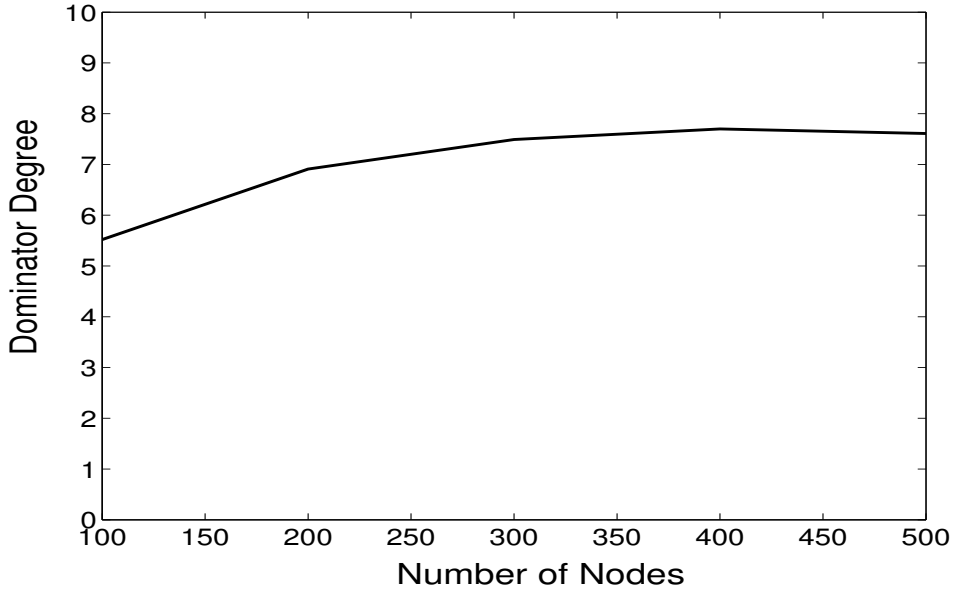


(b) With Inceasing Nodes in a 1500m x750m Area

Figure 4.1: Backbone Scalability on the Dominator Growth Aspect



(a) With 200 Nodes and Increasing Network Area



(b) With Inceasing Nodes in a 1500m x750m Area

Figure 4.2: Backbone Scalability on the Dominator Degree Growth Aspect

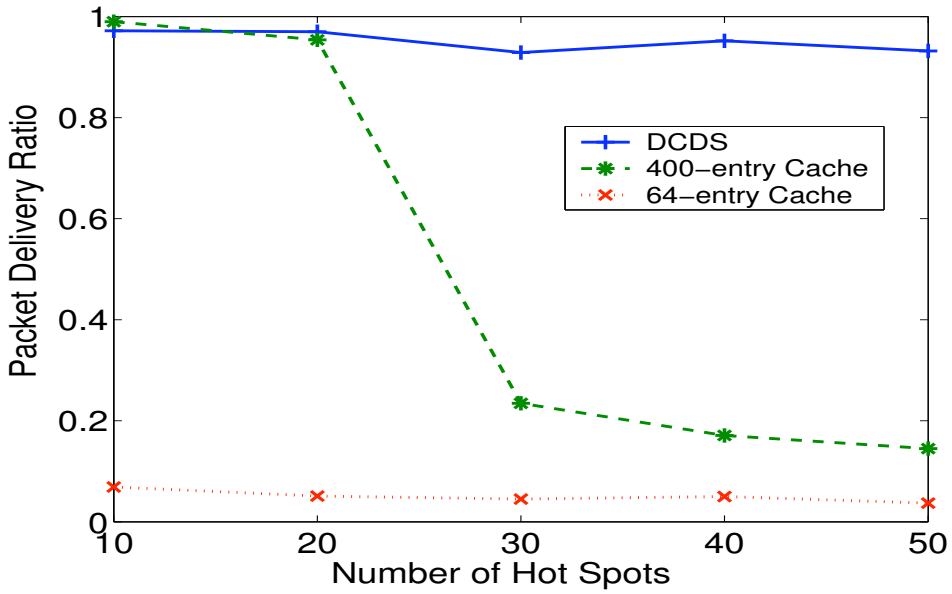
Alzoubi et al. in [16]. However, as mentioned earlier, our experiment is to verify the backbone size and scalability in practice. It is conducted only within static wireless networks where we can easily verify a backbone's correctness, and each data point in Figure 4.1 and Figure 4.2 corresponds to a mean of 30 repeated measurements with different seeds of the random number generator. These different seeds are able to generate different random node distributions in the simulation area.

Figure 4.1 shows how the number of dominators grows as we increase the simulation area and node denstity. It is not surprising that the growth of dominators is linear with the increase of simulation area. As shown in Figure 4.1(a), the number of dominators increases from 6 to 24 while we expand the area by 6 times from $1500m$ x $250m$ to $1500m$ x $1500m$. On the other hand, we can see from Figure 4.1(b) that the number of dominators is barely affected by the factor of node density, given the fixed simulation area and radio range.

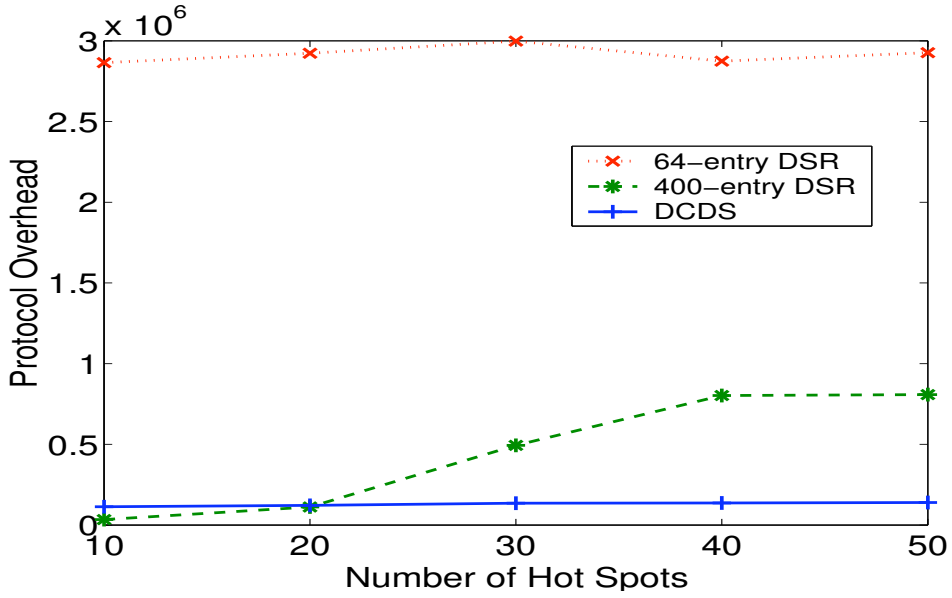
Figure 4.2(a) shows that the inital increase of simulation area from $1500m$ x $250m$ to $1500m$ x $750m$ has an obvious impact on the dominator degree, increasing from 3 to 7. However, as we continue increasing the area, the dominator degree remains constant around 7.3. This is because the initial rectangular areas have a relatively small width, which can easily result in a chain-like backbone topology given the $250m$ radio range. As we increase the width, the backbone turns into a more complicated graph in which each dominator has neighbouring dominators in all directions. However, as soon as the chain-like backbone no longer exists, the increase of simulation area no longer affects the node degree that much even though it still causes the backbone complexity to grow. We can also see from Figure 4.2(b) that the node density has little impact on the dominator degree.

4.3 Varying the Number of Hot Spots

Our second set of simulations give the performance of DCDS as the number of hot spots increases. The network load is fixed to 40 pkts/sec and the CBR lifetime is set



(a) Packet Delivery Ratio



(b) Protocol Overhead

Figure 4.3: Scalability by Varying the Number of Hot Spots

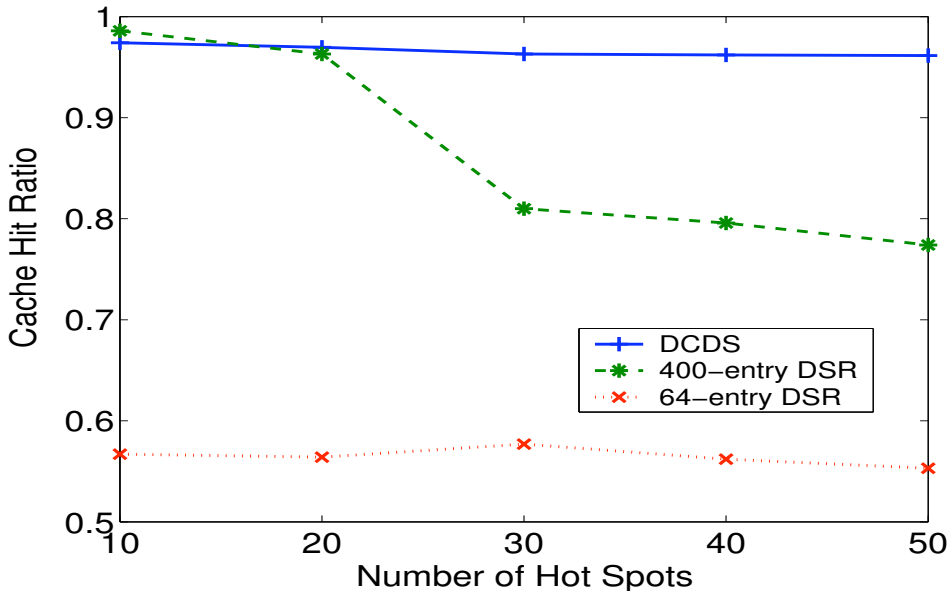


Figure 4.4: Cache Hit Ratio

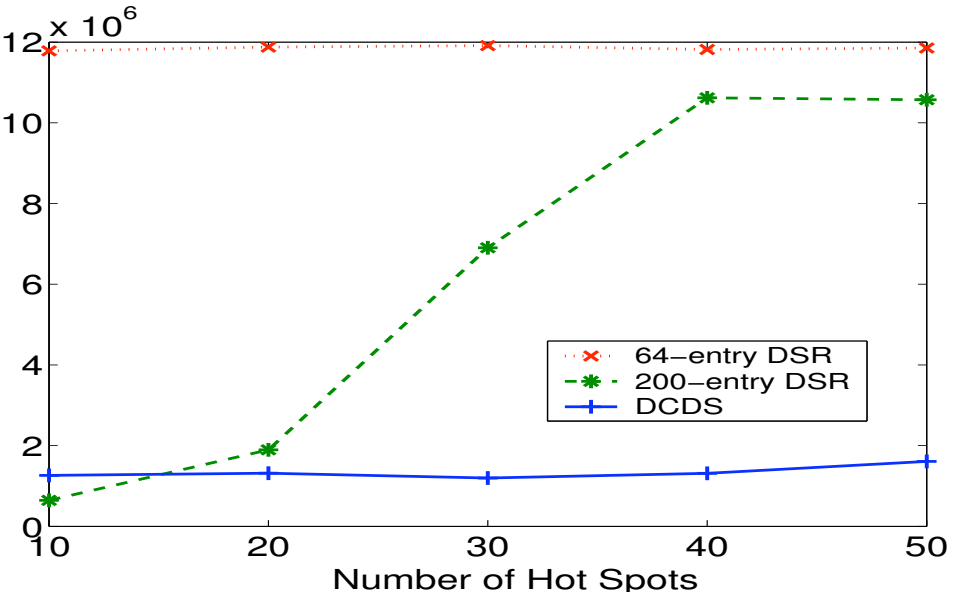


Figure 4.5: Signal Collisions in Radio Layer

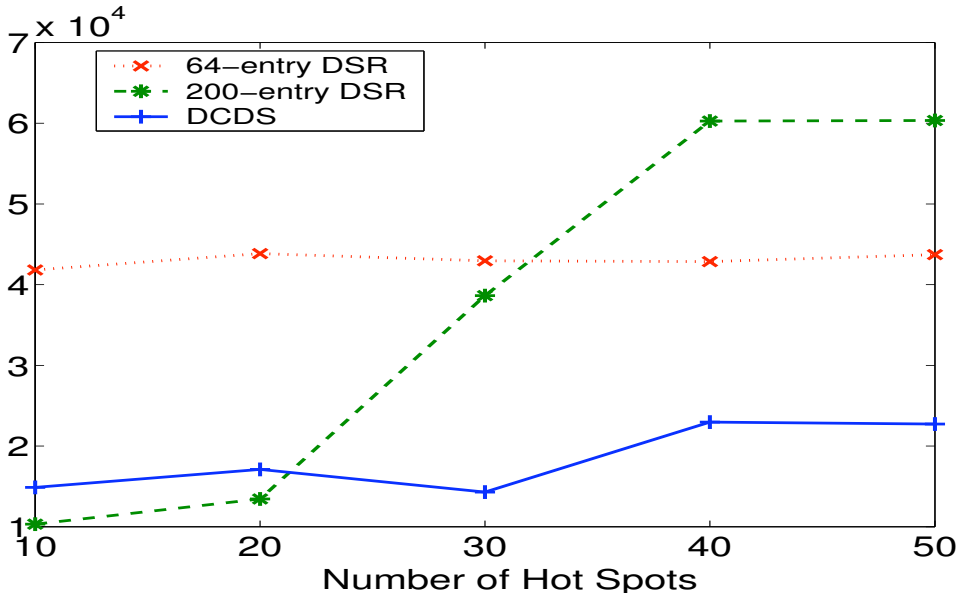


Figure 4.6: Packets Dropped in Mac Layer

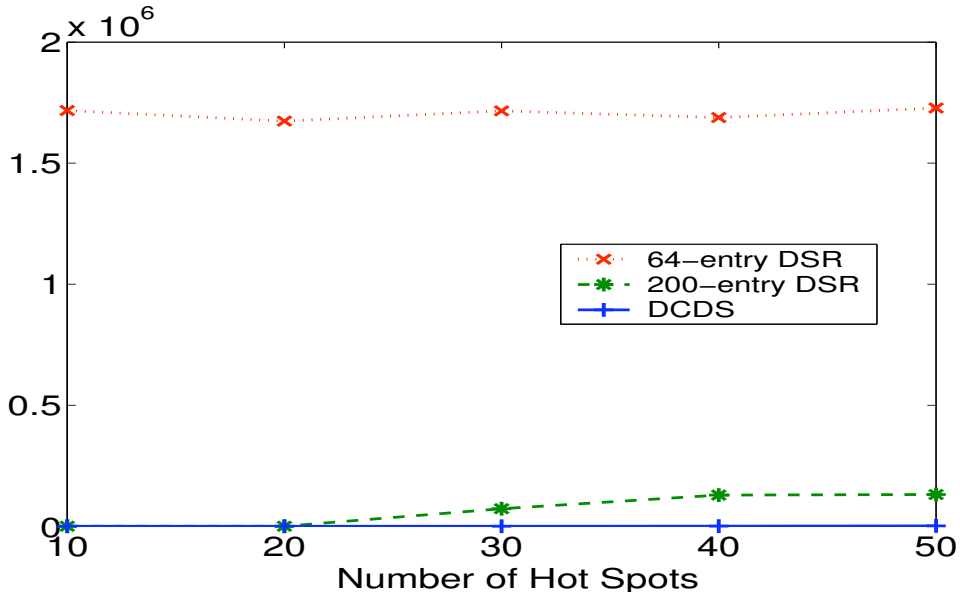


Figure 4.7: Packets Dropped Due to IP Queue Overflow

to 10 seconds. We vary the number of hot spots from 10 to 50, which corresponds to 5% to 25% of the nodes in the network. We compare DCDS with two DSR versions, the standard DSR with a 64-entry routing cache and the modified DSR with a 400-entry routing cache. The 400-entry routing cache is about six times the size of the routing caches in a dominator including BRT, DOT, and DRT.

Figure 4.3 shows the weaknesses of both DSR algorithms. The DSR with a 64-entry routing cache performs consistently the worst among the three. It only provides a 6.9% packet delivery ratio, even when there are as few as 10 hot spots in the network. On the other hand, the DSR with a bigger cache performs very well with a small number of hot spots. However, its performance decreases dramatically as the number of hot spots increases, dropping from nearly 99% to 14.5% as we increase the number from 10 to 50.

The mobility is not a major factor in these results. Instead, DSR's poor performance is due mainly to network congestion. We can see from Figure 4.4 and Figure 4.3(b) that the performance decline is coupled with a corresponding decline in cache hit ratio and increase in protocol overhead. The smaller the number of hot spots in the network and the larger the routing cache, the more likely a node is to locate the destination node in its own routing table and thus avoids expensive discovery broadcast that creates congestion.

DSR with a 400-entry route table performs better than DCDS when there are only 10 hot spots in the network. It also has a smaller protocol overhead since it sends no proactive, periodic control messages. As the number of hot spots increases, however, the 400-entry cache is too small to accommodate the routes to all the destinations. The cache hit ratio thus declines and more route discovery messages are required. These messages are broadcasted in the network and interfere with all the other messages in the airspace. Once the protocol overhead exceeds the limit the network can accommodate, the carrier-sense mechanism and random backoff procedure provided by IEEE 802.11 is no longer effective and the performance declines.

As seen from the figures, the cache hit ratio drops to 81% when we increase the number of hot spots to 30, which results in more protocol overhead and causes the performance to fall off dramatically. The situation is even worse for the DSR with only a 64-entry routing cache.

Bigger caches improve DSR performance, up to a point. To test the limits of this approach, we simulated a version of DSR with an 800-entry cache. This change improved the packet delivery ratio to around 90% for 40 hot spots and 200 nodes. If we increase the number of nodes to 300, however, the packet delivery ratio drops sharply to 8%. DCDS, on the other hand, consistently delivers 95% of its packets no matter how many hot spots exist, as shown in Figure 4.3(a).

Finally notice in Figure 4.4 that DCDS dominators achieve a 97% cache-hit ratio, independent of the number of hot spots. As a result, DCDS performs the fewest route discoveries and consequently has the lowest congestion and highest delivery ratio. To further explain the consequences of network congestion, we show detailed collision and interference information in the radio, MAC and IP layers in Figure 4.5, Figure 4.6 and Figure 4.7 respectively. Figure 4.5 shows the number of signal collisions in the radio layer. This happens when a node receives more than one signal at once, which is well known as the "Hidden Terminal Problem" [35]. Figure 4.6 shows the number of packets dropped in the MAC layer. This occurs when a unicast packet delivery exceeds its retry limit. Moreover as shown in Figure 4.7, 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.

It is interesting to see that DSR with a larger routing cache starts to drop more packets in the MAC layer than DSR with a smaller cache with 30 or more hot spots. There are two reasons for this behaviour. First, DSR with a larger routing cache is able to send more unicast packets into the MAC layer instead of dropping

them in the IP layer. Second, while having larger network topology information can alleviate network congestion, sending the packets via an outdated route causes more delivery failures in the MAC layer due to topology changes.

4.4 Varying Connection Lifetime

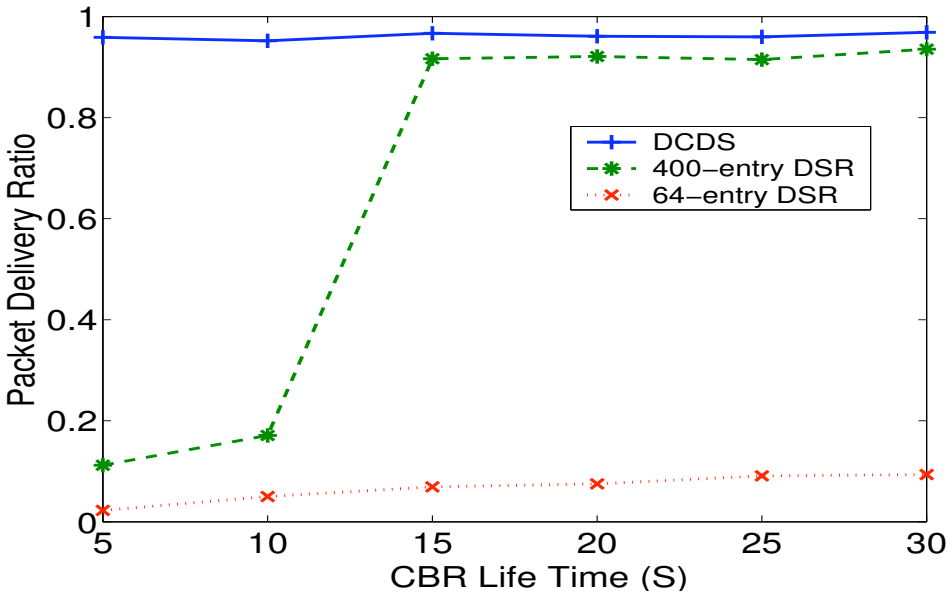
We conduct the third set of simulations to show performance when varying the duration of CBR connections. We set the network load to 40 pkts/sec and fix the number of hot spots to 40 in these simulations.

The results shown in Figure 4.8 clearly match our expectation that the DSR performance deteriorates as the CBR life time decreases. Since the network load is fixed, reducing the CBR lifetime increases the number of CBRs in the simulation. For example, there is a total of 2280 CBRs when the CBR lifetime is 15 seconds; while the number increases to 3400 when we use a 10-second lifetime. As a result, more route discovery and maintenance messages are generated.

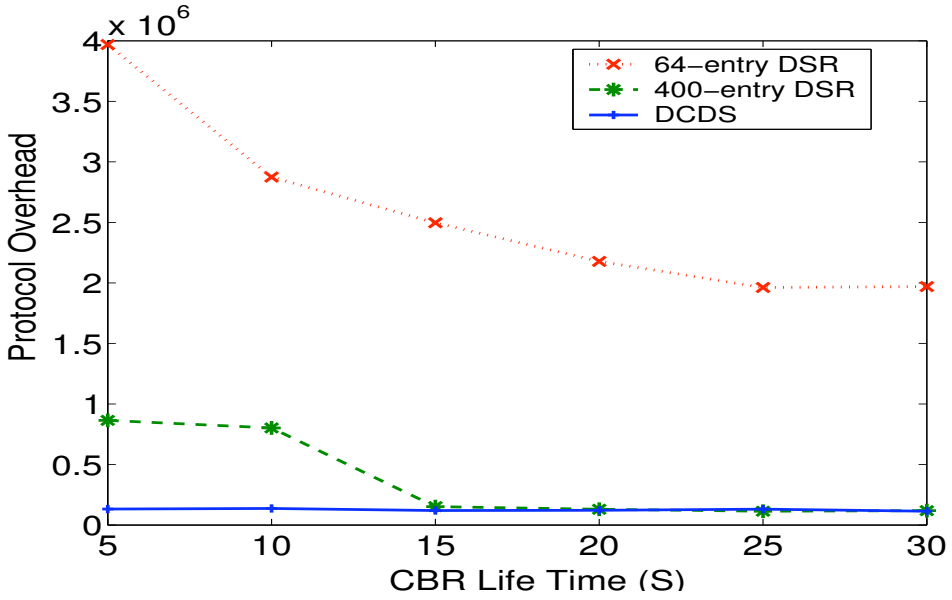
For DSR with a 400-entry routing table, when the CBR lifetime is set to 15 seconds or longer, the total number of data and control messages has not exceeded the network capacity. Therefore, each source node is able to successfully locate the destinations and deliver data packets. However, when the CBR lifetime drops to 10 seconds, the 2Mbps bandwidth is no longer able to handle the increasing network load and thus the network congestion occurs.

Moreover, if route discovery and reply messages are dropped, a source node will generate more subsequent route discovery messages until it receives a route reply or the timer for the reply expires. This, however, causes more protocol overhead and further escalates the network congestion. As shown in Figure 4.8, the packet delivery ratio of DSR with a 400-entry routing cache drops significantly from 92% to 17% as the number of protocol control messages increases from 151685 to 802417, when we reduce the CBR lifetime from 15 seconds to 10 seconds.

Not surprisingly, the DSR with a 64-entry route table still suffers from a



(a) Packet Delivery Ratio



(b) Protocol Overhead

Figure 4.8: Scalability by Varying the CBR Lifetime

severe collision and interference problem and has the worst packet delivery ratio. Nevertheless, as Figure 4.8(b) indicates, we can safely predict that it can continuously improve its performance as the CBR lifetime increases due to the decreasing protocol overhead.

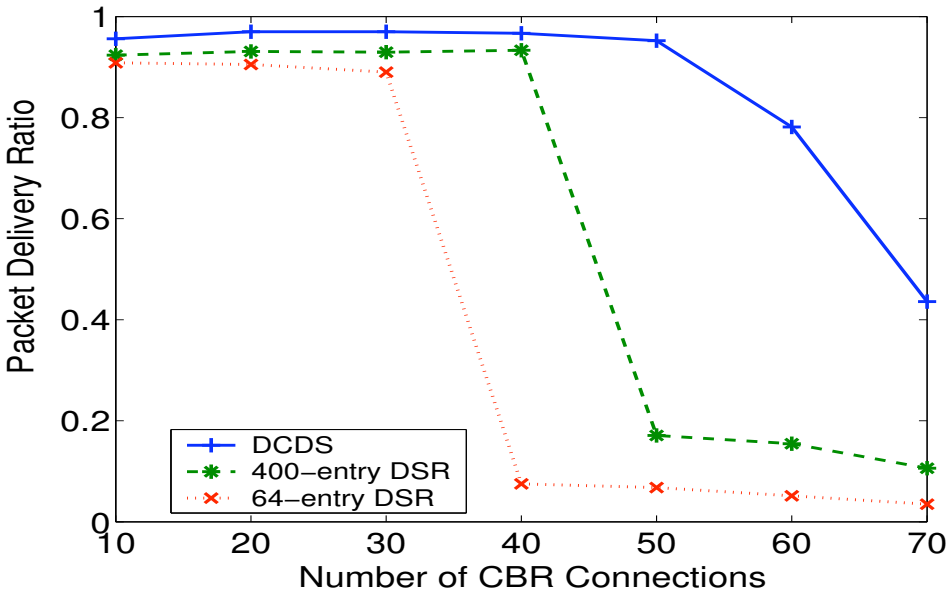
Once again, DCDS shows its advantage in this experiment. For the same reasons discussed in Section 4.3, DCDS avoids the network congestion problem and consistently provides a good packet delivery ratio and generates low protocol overhead. Another interesting observation from Figure 4.8(b) is that, with a 30-second CBR lifetime, DCDS still performs a little better than DSR with a large routing table, even though DSR generates slightly fewer control packets. This is because DCDS handles mobility better than DSR, as described in Section 4.6.

4.5 Varying the Network Load

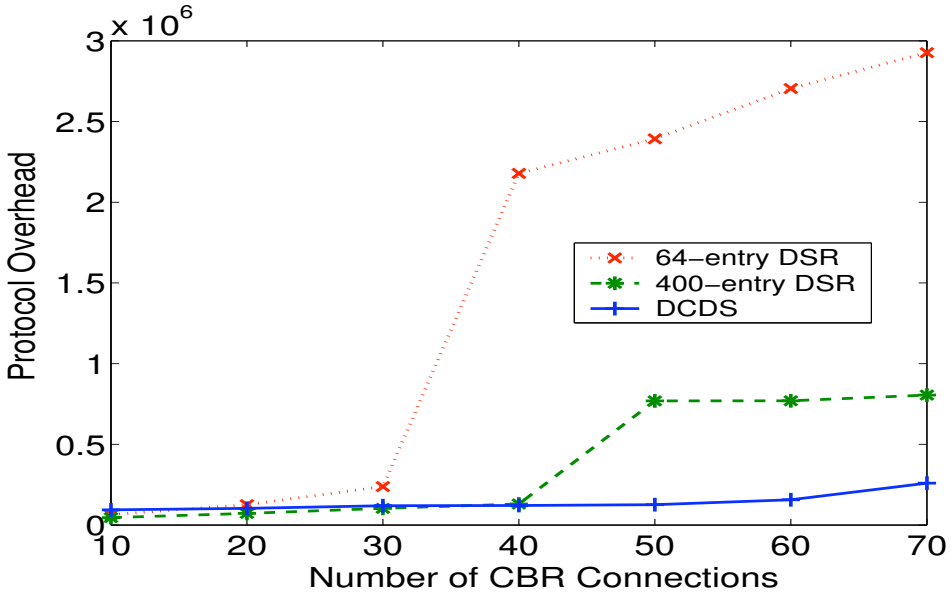
We now evaluate performance as we vary network load. We set the CBR lifetime to 20 seconds and fix the number of hot spots to 40. Since the main focus here is to point out the different ways DCDS and DSR deal with network congestion, we increase the number of CBRs gradually from 10 to 70.

All algorithms perform well when traffic is light. We can see from Figure 4.9, however, that the packet delivery ratios start to decline dramatically with increasing network load. The DSR with a smaller routing table is the first to break, while DCDS is the last to break.

It is obvious that DSR with a larger routing cache outperforms the DSR with a smaller one, since the former results in more route discovery messages. Hence, it saturates the network bandwidth first when the CBR traffic is only 30 pkts/sec. There appear to be two main reasons for DCDS's performance advantage over both DSR versions. First, DCDS's route discovery messages are confined to the backbone and thus each discovery involves fewer message sends. Second, DCDS is able to repair broken paths locally, in the lower-layer backbone protocol, by choosing alternate



(a) Packet Delivery Ratio



(b) Protocol Overhead

Figure 4.9: Scalability by Varying the Number of CBR Connections

connectors to link two dominators. In contrast, DSR's repair is more global, requiring that the route itself be modified and thus involving an error packet sent back to the source node.

All these aspects enable DCDS to generate the lowest overhead and provide the best performance as the number of CBR connections increases. However, Figure 4.9(a) also shows that DCDS starts to break when the number of CBRs increases to 60. As we increase network traffic, the number of packets sent via the backbone will eventually exceed the limit the backbone can handle. Once this point is reached, the local recovery no longer deals with congestion effectively since resending results in more packets and worsens the situation. Moreover, a dominator assumes a backbone link is broken once the local recovery fails, which may cause the backbone to fall apart. A disconnected backbone inevitably leads to more protocol overhead making the network congestion problem even worse, and significantly affects the performance. Figure 4.9(a) shows that packet delivery ratio drops from 95% to 78% as the CBR traffic increases from 50 pkts/sec to 60 pkts/sec, along with a 25% increase of protocol overhead from 125655 pkts to 156210 pkts. The situation unsurprisingly gets worse when we raise the number of CBRs to 70.

4.6 Varying the Mobility

The last set of simulations measures performance with various mobility settings. We fix the CBR lifetime at 10 seconds, and set the maximum speeds to $0m/s$, $5m/s$, $15m/s$ and $20m/s$ respectively in order to simulate different scenarios from a static network to a highly dynamic network with vehicle mobility settings. In the mean time, we keep the minimum speed and the pause time unchanged. To focus our analysis on mobility, the CBR traffic is fixed at only 10 pkts/sec to minimize the impact of network congestion and interference.

As shown in Figure 4.10, DCDS continues to outperform the other two algorithms throughout all the simulations. Its performance drops from almost 100% to

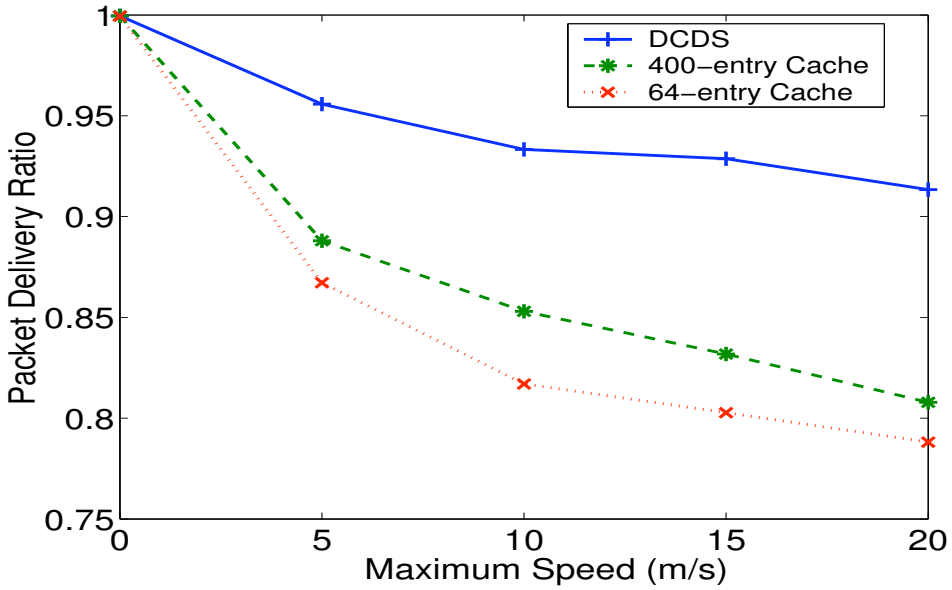


Figure 4.10: Packet Delivery Ratio by Varying Mobility

91% as we increase the maximum speed from $0m/s$ to $20m/s$. On the other hand, the performance of the DSR algorithms, which is the same as DCDS in the static environment, declines at a relatively faster rate, dropping to 80% for the highest mobility settings. Network congestion and interference are not a problem here. As a matter of fact, when setting the maximum mobility to $20m/s$ in our DSR experiments, there was never more than 100 failed data packet transmissions due to network congestion.

Instead, DSR's performance disadvantage is caused by the staleness of its caches. DSR uses aggressive caching in order to achieve low overhead and alleviate network congestion, however this is at the risk of using outdated routes. With short-term CBR connections, a node is not able to keep track of path connectivity after delivering all its data. Therefore, it is likely that, at some point later when the node tries to reuse the path, the path has already broken. This not only potentially causes data packets to drop, but can also pollute the caches of other nodes. Worse,

such a node can propagate the obsolete path to the other source nodes if it receives requests for any node on a broken path.

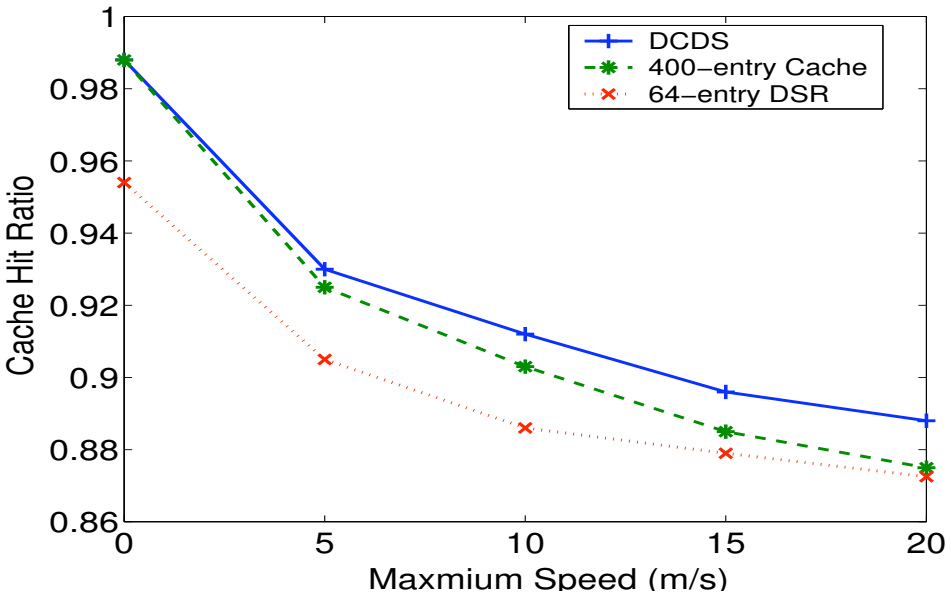
Therefore, we can see from Figure 4.11 that, even though DSR has a relatively high cache hit ratio, source routing with an inaccurate path causes a large number of data packets to drop. The larger the DSR routing cache is, the more data packet transmissions fail. When we increase the maximum speed to $20m/s$, sending 8500 CBR packets actually causes 10207 failed data transmissions in the network. However, the DSR with a bigger cache is still able to provide better performance than the DSR with a smaller cache, because the source node and the intermediate nodes on the path can have more network topology information with a bigger routing table, and thus it can effectively salvage more packets by retrying other alternative paths.

On the other hand, DCDS adopts several optimizations to address this problem. First, a DCDS dominator uses timestamps to keep track of the last hop information in its DRT. For example, a dominator A has a DRT entry (B, X, T_1), which means the dominator B has dominated X since T_1. Later, if A receives a route reply including an entry (B, X, T_2), it compares the T_1 with T_2 and keeps the latest one in its DRT. As a result, it can select the best dominator to reach the destination node.

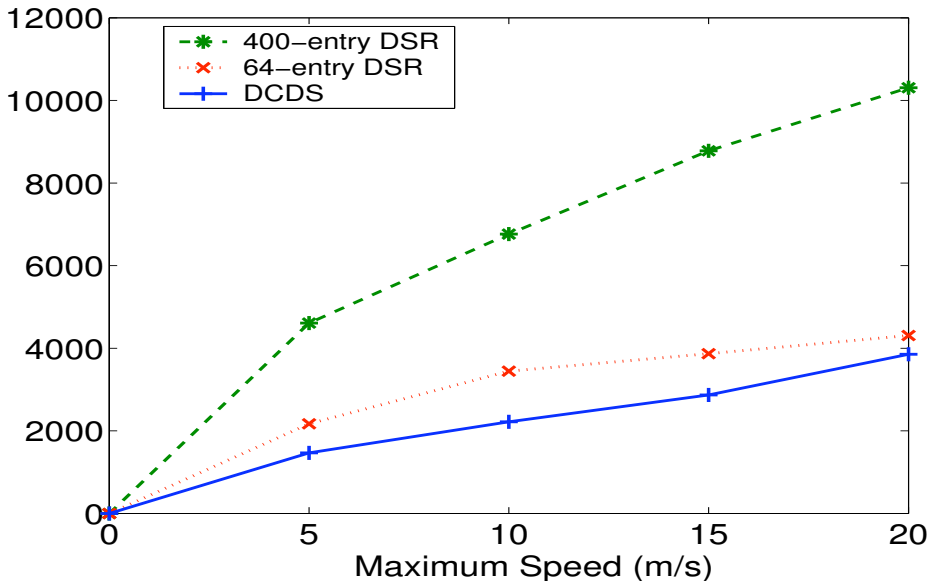
Second, the periodic heartbeat messages and timeouts enable a dominator to maintain a relatively accurate view of its neighbourhood topology. Therefore, as discussed in Section 3.2, the timestamps in the connectivity information list help a dominator to heuristically choose a dominatee which is the most likely to reach its neighbouring dominator. Moreover, it is able to recover a data packet immediately after detecting a transmission failure by using another dominatee.

Finally, the backbone is in use as long as there exists data traffic in the network. Therefore, the dominators are able to detect a broken backbone link quickly, and thus the BRT information is pretty much up-to-date. Furthermore, since the backbone is fully connected and each dominator tries to keep track of all

the dominatees in the network, an intermediate dominator is more likely to salvage a data packet when it fails to transmit a packet to the next dominator.



(a) Cache Hit Ratio



(b) Packet Dropped Due to Mobility

Figure 4.11: Mobility Effects on Caching and Source Routing

Chapter 5

Conclusion and Future Work

5.1 Conclusion

This thesis presents DCDS, a hybrid algorithm that combines proactive clustering and reactive routing for mobile ad hoc networks. The proactive component builds a connected-dominating-set backbone that groups nodes around cluster heads. The reactive component uses techniques similar to DSR, but confines caches to cluster heads and routes to the backbone.

The costs for maintaining the spanning graph are small compared to the benefits. DCDS can discover routes by flooding only the backbone, not the entire network as required by the other algorithms. DCDS can cache routes as aggressively as DSR, while efficiently maintaining cache consistency. It achieves this benefit by using fewer caches and fewer routing nodes than the other algorithms, where any node can act as a routing node. Therefore, in the other algorithms, when a single node moves or fails, it potentially invalidates not only routes to that node but also routes that go through it to other nodes. In DCDS, on the other hand, only a small fraction of nodes are routing nodes and thus most nodes only invalidate routes to themselves when they move or fail.

Our simulation results show that our hybrid approach yields significant performance benefits compared to DSR, the reactive algorithm upon which it is based.

5.2 Future Work

There are three main directions for future work. The first is to further improve upon the scalability of our DCDS routing algorithm. We plan to investigate the performance of a spanning tree backbone routing scheme, where each dominator uses a spanning tree rooted at itself to propagate route discovery messages. This can certainly reduce the number of messages that are unnecessarily forwarded in the backbone. However, it is also challenging to make sure each tree topology is accurate and complete so that a route discovery message can reach all the dominators in the network.

Second, with a shared backbone like DCDS, load balancing becomes an intriguing issue to look into. The dominators in the backbone are going to handle significantly more traffic than the nodes not in the backbone. Therefore, it is also important to come up with a fairness algorithm based on DCDS, which can spread out traffic and share routing responsibilities among nodes in the network.

Another intriguing topic we are looking into is to further investigate the backbone quality and how it depends on the mobility of the nodes. This can be measured in two ways: *backbone coverage* and *backbone connectivity*. Backbone coverage is a measure of how well the backbone can reach all the dominatees in the network; Backbone connectivity is a measure of backbone partitioning, i.e., if there are any dominators that are not connected to each other.

Bibliography

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

[2] R. Ogier, F. L. Templin, B. Bellur, and M. G. Lewis, "Topology broadcast based on reverse-path forwarding," IETF MANET Internet Draft, Nov 2002 (work in progress).

[3] 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.

[4] 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.

[5] 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 (ACM MOBICOM'99)*, 1999, pp. 151–162.

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

[7] F.Bai, N.Sadagopan, and A.Helmy, "Important: a framework to systematically analyze the impact of mobility on performance of routing protocols for adhoc networks," in *Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2003)*, Mar. 2003, pp. 825–835.

[8] ——, "Brics: A building-block approach for analyzing routing protocols in ad hoc networks - a case study of reactive routing protocols," in *USC-CS-TR-02-775*, Nov. 2002.

[9] 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.

[10] P. Johansson, T. Larsson, N. Hedman, and B. Mielczarek, "Routing protocols for mobile ad-hoc networks - a comparative performance analysis," in *Proceedings of the 5th International Conference on Mobile Computing and Networking (ACM MOBICOM'99)*, Tel Aviv, Israel, Aug. 1999, pp. 195–206.

[11] Z. J. Haas and M. R. Pearlman, "The performance of query control schemes for the zone routing protocol," in *SIGCOMM*, 1998, pp. 167–177.

[12] V. Ramasubramanian, Z. J. Haas, and E. G. Sirer, "Sharp: a hybrid adaptive routing protocol for mobile ad hoc networks," in *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing.* ACM Press, 2003, pp. 303–314.

[13] 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.

[14] P.-J. Wan, K. Alzoubi, and O.Frieder, "Distributed construction of connected dominating set in wireless ad hoc networks," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM'02)*, June 2002.

[15] 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.

[16] 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.

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

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

[27] 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.

[28] *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.

[29] Y.-C. Hu and D. B. Johnson, "Caching strategies in on-demand routing protocols for wireless ad hoc networks," in *Proceedings of the Sixth Annual IEEE/ACM International Conference on Mobile Computing and Networking (MobiCom'00)*, Aug. 2000, pp. 231–242.

[30] ——, "Ensuring cache freshness in on-demand ad hoc network routing protocols," in *Proceedings of the POMC 2002 Workshop on Principles of Mobile Computing*, Toulouse, France, Oct. 2002, pp. 25–30.

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

[32] 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.

[33] CMU Monarch Group, "The CMU Monarch Project's Wireless and Mobility Extensions to NS," Aug. 1998. [Online]. Available: http://citeseer.nj.nec.com/180061.html

[34] M. Takai, J. Martin, and R. Bagrodia, "Effects of wireless physical layer modeling in mobile ad hoc networks," in *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking and computing (MobiHoc'01)*, Long Beach, CA, 2001, pp. 87–94.

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