Improving Backbone Routing for Transient Communication in Mobile Ad Hoc Networks

Kan Cai, Michael J. Feeley, and Norman C. Hutchinson
Department of Computer Science
University of British Columbia
Vancouver, BC, Canada, V6T 1Z4
{kcai, feeley, norm}@cs.ubc.ca

***Abstract*— Multi-hop routing in mobile ad hoc networks is challenging due to node mobility, low power, constrained bandwidth and limited radio range. Previous work has shown that reactive flat algorithms outperform proactive algorithms under high mobility. Hybrid and hierarchical approaches further improve performance by reducing the range and frequency of broadcast route discovery messages.**

Current algorithms, however, perform poorly when dealing with spontaneous, transient communication. This traffic pattern generates route discoveries much more frequently than long-term communication and thus causes the reactive component of flat routing algorithms to flood the network, triggering broadcast storms that render the network temporally useless. Similarly, the increased number of route discoveries also makes the backbone a bottleneck for the hierarchical algorithms.

This paper describes the design of a backbone routing scheme, called DCDS, that handles transient traffic well. Like other backbone routing algorithms, it uses proactive components to group nodes into clusters and construct a backbone; it then uses a reactive protocol to route packets over the backbone. The three key novel features of our algorithm are: (1) it uses piggybacking to prefetch multiple routes in response to a single request, (2) it timestamps cache entries to provide a heurstic for flushing out-of-date routes from caches and (3) it uses proactive, local information to adapt to and recovery from some failures without changing globally-cached routes or dropping packets. We evaluate DCDS using Glomosim to simulate it and three other algorithms, two variants of DSR and a model hierarchical algorithm, for comparison. Our results show that DCDS performs substantially better than these other algorithms for the workloads we studied.

I. Introduction

The ad-hoc mode of IEEE 802.11 (i.e., WiFi) allows nodes to form single-hop networks without the aid of wired infrastructure [1]. A variety of higher-level protocols have been proposed for building multi-hop networks that route packets from node to node using 802.11 ad-hoc mode. 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 an access point.

Multi-hop, ad-hoc, wireless protocols can be roughly be divided into two categories — *flat* and *hierarchical* — based on the way they organize routing information. Flat routing algorithms require each node to discover and maintain routing information for itself, while hierarchical approaches organize nodes into clusters and then delegate routing responsibilities to a subset of nodes, the cluster heads.

Another way to classify protocols is as either *proactive* or *reactive*, based on the way they collect routing information. Proactive, flat protocols such as DSDV [2] use periodic control messages to maintain up-to-date routing information and are ready for sending a packet anywhere at anytime. However, they impose a fixed message overhead for control messages and fail to adapt quickly to topology changes.

Reactive, flat protocols such as DSR [3], on the other hand, discover and maintain a route only when needed and thus have no fixed overhead and can adapt quickly to node mobility and failure. However, they suffer from severe network congestion when route discovery, which requires global broadcast, is too frequent.

Hybrid, flat protocols combine some proactive and some reactive features in an attempt to exploit the best of each. ZRP [4], for example, maintains local routing information proactively, but only builds routes to remote nodes on demand. In practice, these algorithms work well when most communication is local, but suffer the same congestion problems when discovery of remote nodes is too frequent.

In contrast to *flat* protocols, *hierarchical* protocols minimize overhead by aggregating routing information and reducing the range and frequency of route-discovery broadcasts. Protocols such as HSR [5] and HierLS [6], for example, build multi-level, multi-hop clusters and then proactively maintain global routes among the clusters. Other *hierarchical* approaches such as DSRCEDAR [7] and CBRP [8] proactively maintain a spanning backbone and then use the backbone to perform route-discovery broadcasts more efficiently. Routing itself is reactive, using an algorithm such as DSR, but modified to perform backbone-based route discovery.

The comparitive performance of these algorithms varies, depending on workload. If nodes tend to communicate mostly with closeby nodes, ZRP and hierarchical protocols benefit. If nodes tend to communicate with a small number of other nodes, DSR benefits. One thing that all of the algorithms have in common is that they work best when node discovery is infrequent. Therefore, all the previous work focuses on performance evaluation in scenarios where unique connections are few and long lived using continuous end-to-end traffic.

However, only studying fixed-endpoint, long-term traffic in ad hoc networks is questionable, at least incomplete. Previous analyses of publich wireless networks in both academic and corporate environment [9, 10] have shown that users are often passive and network traffic is bursty. The wireless sessions are usually short lived and the long-term connections are idle most of the time. Later studies on PDA users [11, 12] further showed that median user session duration was only 5-6 minutes and median AP session was only 1.8 minutes. Moreover, applications such as web browsers and messengers are inherently bursty. Even in a single session, they might mostly remain inactive and generate packets destined to *different destinations for a short period*, a few seconds for example.

We suspect that traffic in mobile ad hoc wireless networks would follow the same pattern even though no such trace is available. This traffic pattern generates route discoveries and reparations much more frequently, due to multiple destinations and outdated routing information, than the long-term fixed-endpoint communications, and thus easily causes the reactive component of flat routing algorithms to flood the network. These network-wide broadcast messages can clog the shared airspace and deteriorate the signal interference and network congestion, easily triggering broadcast storms that render the network temporally useless.

Likewise, the increasing number of route discoveries also makes the backbone the bottleneck for the hierarchical algorithms since the backbone handles all the route discovery messages. It could eventually make enough congestion problem for the backbone to fall apart.

In this paper we described a new protocol, called DCDS, that we have designed to work well when connections are short lived and when communication is frequent and not confined to a local neighbourhood. Our evaluation shows that existing algorithms perform poorly for this class of workload, due primarily to the high cost of frequent route discovery. For example, one of our reported simulation scenarios is a network of moderate density and mobility (i.e., 200 nodes in a $1500m$ x $750m$ area and with 1-5 m/s random-waypoint mobility) in which 20% of nodes are designated to be *hot spots*. When senders are chosen randomly so that 20% are sending at a time and each selects a random hot spot and then sends packets at 1 packet/s for 10 seconds, DSR delivers only 18% of these packets successfully and a model hierarchical algorithm only 63%, but DCDS delivers 96% of packets and does so with lower overhead than the other two.

DCDS is a hybrid, hierarchical routing protocol similar to other protocols such as DSRCEDAR and CBRP, but with three key differences. First, DCDS piggybacks additional information on each route-discovery reply message to prefetch routes for multiple nodes for each discovery request. Second, DCDS uses timestamps on route-cache entries to keep cached routing information current. Third, DCDS uses proactive, local route adaptation and recovery to dynamically change inter-backbone-node links without changing globally caches routes or dropping packets.

This paper describes the design of DCDS and evaluates the protocol using simulation, comparing it to DSR and an alternative hierarchical protocol, DSRX, modeled after DSRCEDAR. We show that DCDS outperforms the others by delivering a higher portion of packets successfully and doing so with lower packet overhead. We also show the extent to which each of DCDS's three features contribute to this performance improvement.

II. Background and Related Work

Research on clustering and backbone routing in mobile ad hoc networks has a long history, originating from radio networks [13, 14]. Generally speaking, it organizes the nodes into clusters and designates the routing function to a subset of nodes, usually the cluster heads. In this section, we briefly discuss the advantages and drawbacks of backbone routing schemes, and then describe other work related to our approach.

A. Pros and Cons of Backbone Routing

The principle advantage of previous work on backbone routing is that confining route-discovery to the backbone greatly reduces the congestion caused by route discovery messages, which in flat algorithms flood the entire network. Another advantage is that backbone routing is more resilient to node failure and mobility for two main reasons. First, most nodes are non-routing nodes and thus their failure invalidates only routes for which they are an endpoint. In a flat algorithm, on the other hand, every node is a routing node and thus a single failure invalidates the potentially-many routes that travel *through* the failed node. Confining routes to the backbone, on the other hand, means that the failure of a backbone node has greater impact. This disadvantage can be offset by the fact that a single backbone repair can potentially salvage all of these failed routes in one step. The second way that backbones can be more resilient to failure is that they can confine route caches to backbone nodes. Flat algorithms cache routing information at every node and thus globally cache many more copies of each route, and thus a single failure invalidates more routes, each invalidation coming at the cost of additional overhead and, often, undelivered data.

The main drawbacks of backbone routing are unfairness, backbone bottlenecks and route inefficiency. Since much of the network load is focused on nodes in the backbone, if nodes are equal peers, then a node pays a high price if it is selected to be part of backbone. Therefore, it is desirable to rotate the role of cluster head among the nodes in the network [15, 16]. Backbone routing also does not necessarily avoid congestion when the packets could otherwise be delivered using non-interfering paths [16]. Intuitively, forwarding all the traffic via a backbone can easily make the backbone the bottleneck in the network. Finally, backbone routing uses sub-optimal routes, requiring more hops than necessary [17]. To some extent, these problems are the price paid for the efficiencies the backbone delivers in other ways, primarily reducing congestion caused by route discovery.

B. Related Work

Hierarchical algorithms differ in the way they use the backbone to route packets. Some closely integrate routing with the backbone construction algorithm itself, using proactivity in aspects of both, while others layer a reactive algorithm such as DSR, or sometimes a hybrid algorithm, on top of the backbone in a more modular fashion.

Hierarchical link-state protocols such as HSR [5] and HierLS [6] follow the more integrated approach. They build multi-level and multi-hop clusters, and then proactively maintain routing information to all the other nodes in the network. This integrated approach is complex and can thus be hard to implement.

Another set of integrated approaches are the spine-based algorithms from R. Sivakumar et al. [18–20]. Compared to HSR and HierLS, the spine approach is simpler, because it uses only two layers in the routing hierarchy. In addition, shallower hierarchies suffer less from the problem of route inefficiency that occurs when long-distance routes are forced to travel through nodes high in the hierarchy. One of the spine papers [19] provides a formal description of an algorithm that uses a local-recovery scheme similar to ours, but does not go beyond this formal description. As well, due to the more formal nature of this work, they assume that the radio network provides a reliable broadcast mechanism, which 802.11 does not, and without which routing caches will be incomplete or inaccurate.

In any case, the main potential drawback of integrated approaches such as these is that by making the routing protocol proactive, because it is integrated with the proactive backbone, there is a high cost to maintaining routing structures, particularly when nodes fail or move.

The alternative to the integrated approach is to layer a reactive algorithm on top of the backbone; this is the approach that we follow with DCDS. There are four main algorithms, other than ours, that following the layered scheme.

First, K. Xu et al. [21, 22] propose a two-level, hierarchical, clustering algorithm, called *mobile backbone*. It uses distinguished backbone nodes with more-powerful radios that communicate directly with each other. This direct communication simplifies the routing problem significantly compared to the environment we target.

Second, M. Jiang et al. [8] describe a DSR-like routing algorithm on top of a backbone, called CBRP. All nodes send periodic, one-hop broadcast *hello messages*. These messages are used to elect cluster heads and maintain cluster-membership lists. Cluster heads also use these messages to build inter-cluster connections using gateway nodes. CBRP uses a complex variant of DSR to route packets on this backbone. Like DSR, its routing is entirely reactive; there are no periodic exchanges of routing information among cluster heads. Also similar to DSR, if a cluster head fails to resolve a destination node in its routing cache or neighbour list or if it detects a broken route, it floods the backbone using a combination of broadcast and unicast messages.

Third, P. Sinha et al. [23] propose a core-based routing algorithm, called CEDAR, which includes various features design to support QoS routing. Similar to CBRP and our algorithm, each core node establishes tunnels with its neighbouring core nodes via three-hop broadcast messages. Unlike other algorithms, CEDAR, modifies the MAC layer and uses promiscuous mode to limit core-broadcast message propagation. Also, it is able to compute a "shortest-widest" path from the source to the destination, by proactively propagating topology changes and link statistics in the core using waves.

Fourth, P. Sinha et al. subsequently improved CEDAR by reducing its core broadcast overhead [7]. Unlike CEDAR, routing is performed by a standard, flat protocol layered on top of the core. DSRCEDAR uses DSR for routing and AODVCEDAR uses AODV [24]. Their simulation results show that the addition of the backbone improves the performance of both of these standard, flat algorithms. Similar to CEDAR these algorithms rely on a modified MAC layer and on the use of promiscuous mode.

In many ways our work builds on earlier systems such as CBRP and DSRCEDAR that layer DSR or a similar algorithms on top of a backbone. Our contribution is to extend the basic, reactive routing algorithm to better handle transient communication and the inherent potential drawbacks of backbones.

III. DCDS Algorithm

DCDS is partly proactive and partly reactive. Its proactive component acts to maintain a single spanning graph for the network. Routing, on the other hand, is performed reactively in a manner similar to DSR [3], but where routes are confined to follow the spanning-graph backbone.

The backbone is constructed using a variant of the Message-Optimal Connected Dominating Set Algorithm [25] that we modified to work incrementally and to be resilient to communication failures. The algorithm selects certain nodes to act as cluster heads, called *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.

What makes DCDS unique compared to other hybrid, backbone routing schemes such as DSRCEDAR [7] and CBRP [8] is the way it uses the backbone to improve the route-cache effectiveness and thus reduce the frequency of route discovery. It does this in three main ways. First, it uses piggybacking to prefetch routing information into caches. Second, it uses timestamps to flush out-of-date routes from caches *without using any time synchronization algorithms*. Third, it uses proactively maintained neighbourhood information maintained by each dominator to adapt to and to recover from changes in inter-backbone-node connectivity without invalidating globally cached routes or dropping packets.

The key feature of DCDS that enables these optimizations is the way it organizes routing information, using a three-tiered structure. The first two tiers are route caches stored at dominators: one cache records backbone topology and the other records dominatee-dominator relationships. A source dominator determines a route by using one cache to find the destination's dominator node and the other to plan a backbone route to that dominator. Both of these are maintained reactively and are thus somewhat incomplete and inaccurate.

The third tier of the routing strategy is the list of connections each dominator *proactively* maintains to other dominators in its three-hop neighbourhood. When forwarding packets, a dominator uses this information to choose a local path to the next dominator on the route. DCDS is thus able to adapt locally to changes in dominator connectivity, without dropping data or invalidating routing information cached at other dominators. Global route invalidation is necessary only when a dominator itself fails or when all connection between a pair of dominators is lost.

The remainder of this section describes the unique features of DCDS in detail. We describe the design in two parts. First we describe the lower-level, proactive backbone maintenance algorithm. Then we describe the higher-level, reactive routing algorithm.

A. *Proactive Backbone Construction*

The basic operation of the maintenance protocol is to group nodes into clusters around a cluster-head, called 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, while dominatees also act as *connectors* that link dominators to each other. Dominators store a list of in-radio-range dominatees in the *dominatee ownership table* (DOT), and each DOT entry is timestamped when added. Dominators also store a *connectivity list* containing paths to other dominators that are two or 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. 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 heartbeat message to every node in its radio range. This message is timestamped and contains the dominators ID. When a dominator node receives a DOMINATOR heartbeat with an ID lower than its own, it changes its state to dominatee. When a dominatee receives a DOMINATOR heartbeat, it records the dominator in its local list and timestamps that entry with that dominator's timestamp.

Each dominatee periodically broadcasts a DOMINATEE heartbeat message to every node in its radio range. This heartbeat message is timestamped and includes the dominatee's ID, a list of dominators within two hops of the node, and a vector timestamp indicating the freshness of the dominator list. The dominator list is initially empty. The vector timestamp is updated transitively by dominator heartbeat messages.

When a dominator receives a DOMINATEE heartbeat, it does two things. First, it adds the dominatee to its DOT, if it is not there, and it timestamps the DOT entry with that dominatee's timestamp *and* the current time. Then, it adds each dominator listed in the heartbeat message, along with its timestamp, to its conectivity list designating the dominatee as the first-hop connector for each. For three-hop-away dominators, the second hop node is not listed in the dominator's connectivity list. Instead, this second-hop node is determined by the first-hop dominatee from its own connectivity list.

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

If a dominatee node fails to receive any DOMINATOR messages, after a timeout period, it initiates a process to determine whether it should become a dominator. The failure to receive a DOMINATOR heartbeat, however, is an insufficiently strong indicator that there are no dominators in range. The problem is that in 802.11 broadcast messages like the heartbeats have lower priority than unicast messages. They are thus frequently drowned out during periods of sustained congestion.

As a result, before a dominatee declares itself a dominator, it 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 dominatee change its state to dominator and broadcast a DOMINATOR message.

B. *Reactive Backbone Routing*

We now describe the reactive backbone routing protocol that delivers payload packets from source nodes to their targets. To send a message, a 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, its upper layer of the backbone routing component adds this sequence of dominator-node IDs to the packet and hands the packet to the lower layer for delivery to the first dominator on the path. At each dominator, the lower layer upcalls the upper routing layer so that the upper layer can learn or update the backbone path, if needed, 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 first give an overview of DCDS's caching and base routing scheme. Following this is a detailed discussion of the three key, novel features of DCDS: cache-entry timestamping, piggybacked, route-cache prefetching and proactive, local route adaptation and recovery.

1) Routing Caching: The two routing caches stored on dominator nodes are the *dominatee routing table*, DRT, which caches dominatee-dominator pairings, and the *backbone routing table*, BRT, which caches backbone-topology in the form of a list of connected dominators. Dominators maintain their DRT and BRT caches reactively using the content of messages they receive, either as the target or as a routing node.

The DRT is updated by route-discovery reply messages, which typically include multiple dominatee-dominator pairings, due to the piggybacking feature described in Section III-B.4. When a dominator receives a reply — either for itself or one that it forwards to another dominator — it adds the pairing information in the message to its DRT.

The BRT is updated by every packet a dominator receives. Typically, the BRT accurately captures backbone topology and rapidly adapts to backbone changes. These desirable properties derive from the fact that every packet lists the path the packet traversed to arrive at the dominator.

2) Base Routing Scheme: Similar to DSRCEDAR, DCDS lays a DSR-like routing protocol on top of the backbone. It uses this protocol to discover and to maintain backbone routes using three types of control messages: *route discovery*, *route reply* and *route nack*.

A dominator triggers the route discovery procedure when it is unable to resolve a path to the destination node encoded in a packet it receives from one of its dominatees, either because the target is missing from its DRT or because it has no BRT route to the target's dominator. In either case, the dominator sends a unicast discovery message to each of its backbone neighbours.

When another dominator receives a discovery message, it first checks its DOT to determine whether it has the target node in radio range. If not, it checks its DRT and BRT to determine whether it caches a route to the target. If all of these checks fail, it adds itself to the *path-traveled* list in the message's header and forwards the message to all of its backbone neighbours that are not yet listed in the path-traveled list.

When a dominator locates the target, it sends a reply message by reversing the path-traveled list in the request message. This path, possibly combined with BRT information at the dominator, comprise the backbone path from requesting dominator to target dominator.

Whenever a dominator is unable to forward a packet to a neighbouring dominator, it performs the following error-recovery procedure. First, it deletes the backbone link between itself and the neighbouring dominator from its BRT. Then, it sends a *route-nack* message over the reverse path-traveled route back to packet's source dominator. Each dominator that receives the route-nack message removes the failed link from its BRT. Finally, for data packets, it attempts to salvage the packet by looking for an alternate backbone path to the target in its BRT.

3) Cache timestamping: A common problem with reactive caches used by algorithms like DCDS and DSR is detecting routes that become invalid when nodes move or fail. The caches typically hold multiple paths to each target, some that are invalidated by a particular failure and some that aren't, but it can be costly to determine which is which. In DSR, for example, a route discovery fills the requester's cache with all of the available routes to the target node. When a failure invalidates some of these routes, the node is only informed when it uses an invalid route, forwards a route-nack packet to another dominator that has done so, or receives a route-discovery message that piggybacks this failure notification. As a result, a moving or failed node causes many other nodes to route packets erroneously thus increasing overhead and decreasing delivery reliability.

In DCDS this problem primarily affects the accuracy of the DRT caches that record node location by pairing nodes with their nearby dominators. The DRT attacks this problem by using timestamps to estimate the liveliness of cached pairings and then using this liveliness as a heuristic to predict accuracy.

The DRT uses timestamps to ensure that it stores at most one dominator pairing for each dominatee. Recall that dominators get new DRT information as a side effect of receiving route-discovery reply messages. Each dominator-dominatee paring in these messages originates from a DOT at a dominator that has had the paired dominatee in radio range. That dominator timestamps its DOT entries when it adds them as the result of receiving a message from a previously unknown dominatee. Thus, when a remote dominator compares two possible dominator pairings for a dominatee, the pairing with the most recent timestamp is a good estimate of the current location of the dominatee.

The use of cache timestamping is similar to that of sequence numbers in AODV in that it helps to broadcast the latest topology changes. It is also worth noting that, just as sequence number, timestamp is generated by each node based on its own clock and thus it does not require any local or global time synchronization algorithms such as RBS [26]. However, the way DCDS stores, propagates and uses timestamp quite differs from that AODV uses sequence number.

Initially, a dominatee's timestamp is only locally stored in its own dominator's DOT entry. It is later propagated to the other dominators in the backbone instead of the rest of the network. As later discussed in Section III-B.4, these DOT topologies is only piggybacked in route reply messages, not exceeding the IP MTU, and therefore it does not entail constant overhead.

Dominators also use DRT timestamps when they forward packets along the backbone. Each forwarding dominator checks its DRT to see whether it has a *newer* entry for the target than reflected in the current route. If so, it updates the packets route accordingly before forwarding the packet toward this updated location.

Timestamps are also used by the lower-level routing mechanisms that connects neighbouring dominators to each other. Often a pair of dominators have many potential two- and three-hop paths that connect them. Connectivity lists that describe these paths are timestamped by heartbeat messages. When a dominator forwards a packet, it picks the connector with the most recent timestamp.

4) Piggybacked, Cache Prefetching: The second important optimization in DCDS is that route-reply messages are padded to piggyback route information for multiple target nodes. This optimization is important due to the high cost of route discovery. In other algorithms, a route-discovery message typically resolves a route to a single remote node. In DCDS, the two-part route caching scheme enables route information to be compacted in such a way that a single reply can resolve routes to many distinct nodes, at the cost of a small marginal increase reply-packet size. DCDS exploits this route-compaction scheme to prefetch multiple routes into the DRT caches of nodes that receive route reply messages.

To do this, every dominator maintains a vector timestamp that records the last DOT update it received from every other dominator. It includes this vector timestamp in route-discovery messages it originates. When a request arrives at the target dominator, its entry in the request's vector timestamp indicates which of its DOT have not yet been cached by the source. The target dominator selects as many DOT entries with timestamps after this time as will fit in the reply message and sends the reply, along with the request-messsage's vector timestamp. Every node on the reply path repeats this process using their own entry in the message's vector timestamp until the packet reaches its maximum size (IP MTU), if it does. Each also extracts entries from the packet to add to its own DRT.

5) Route Adaptation and Recovery: The final key feature of DCDS routing is the way it corrects for local failures.

Recall that backbone paths listed in the BRT consist of only dominators. 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.

Moreover, with the help of timestamps, a dominator can always take advantage of the latest topology information.

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 the connector node chosen is in range of the dominator. 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 their 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 a statically-defined retry limit is reached or when all available connections have been deleted is the error reported to the upper routing-level protocol. This error initiates a route nack packet that is sent back to the source dominator, and triggers an attempt to salvage the packet at that level if alternate backbone routes exist to the packet's ultimate destination.

IV. Evaluation

This section describes simulation results taken using Glomosim [27], a scalable simulator with accurate physical-layer and radio-propagation models. We set Glomosim parameters as follows: per-node bandwidth is 2 Mb/s, radio frequency is 2.4 GHz and transmission range is 250 m.

The evaluations are conducted with a total of 200 nodes randomly distributed in an area of $1500m$ x $750m$. We choose an area that is three times larger than previous work [28, 29] in order to avoid the formation of chain-like backbones (i.e., where the backbone node degree is low), which would tend to favour backbone approaches such as ours. We verified that chain-like backbones were typically not created by evaluating the backbone topology produced when nodes are not mobile. We varied the Glomosim seed value for a total of 10 trials. The resulting networks had an average of 14.5 dominators (range: 13–17; std. dev. 1.2) with an average node degree of 7 two- or three-hop neighbouring dominators (range: 3–11; std. dev. 2).

Each simulation lasts 910 seconds. We eliminate startup and shutdown effects by waiting 50 seconds before opening the first CBR connection and stopping measurements 10 seconds before the end of the simulation. We perturb the start time of each CBR source by up to 10 seconds to reduce the probability that synchronization among senders causes unnatural congestion.

We use Random Waypoint [3] to model mobility. Using this model, each node randomly chooses a destination and moves towards it with a velocity chosen randomly from $[V_{min}, V_{max}]$. 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 5.0 m/s in most of the simulations. However, in Section IV-F, we examine DCDS performance for a variety of faster mobility settings.

TABLE I
DCDS Protocol Settings

DOMINATOR heartbeat interval	0.5 s
DOMINATEE heartbeat interval	5 s
dominator timeout	5 s
dominatee timeout	30 s
local-recovery retry limit	4
packet-salvage retry limit	2
time to hold packet awaiting routes	30 s
time to hold packet after forwarding	5 s
dominator-AYA response timeout	1 s

We adopt a multi-destination, constant-bit-rate (CBR) traffic pattern similar to that used in the SHARP paper [30]. 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 fixed rate of one packet per second. We vary network load by changing the number of concurrent CBR connections.

Finally, all of the data reported in this section is the mean of three trials; the variance between the trials was low. The main metrics we use are packet delivery ratio (PDR) and message overhead. PDR, a common metric, gives the fraction of packets successfully delivered compared to the number sent.

A. The Protocols

We compare DCDS to three other protocols, two versions of DSR that we call SDSR and BDSR and a model hierarchical protocol we built called DSRX.

SDSR is the standard version of the protocol reported in the literature [28, 29]. To ensure our comparison was as faithful as possible to previously reported DSR results, we imported the $ns2$ DSR code from the Monarch project [31] and modified it to work with Glomosim. We verified that this implementation closely matches the $ns2$ version when physical values are set as suggested by [32].

BDSR is the same as SDSR in all respects except that nodes use bigger routing caches: 200 entries instead of 64. We use BDSR because our study shows that the bigger cache improves DSR performance in certain cases. BDSR's big cache, however, causes it to under-perform SDSR when nodes are very mobile, as we show in Section 4.6.

DSRX is a protocol we implemented as a model for class of hierarchical protocols that are similar to DCDS (e.g., DSRCEDAR). It uses the same backbone algorithm as DCDS, but with a standard implementation of DSR built on top of it. We use DSRX to illustrate the benefits of the three key features of our algorithm — piggyback prefetching, cache timestamping and local, proactive recovery — none of which are part of standard DSR nor DSRX. Instead, DSRX uses DSR's standard packet salvaging scheme, which, in fact, is more aggressive than ours.

We compared DSRX to the reported performance of

DSRCEDAR in an effort to determine the extent to which our DSRX results could be generalized. Similar to other related work, however, the published results for DSRCEDAR are for longer-term connections than those we focus on in this paper (i.e., roughly 800s) [7]. For this type of connection, our simulations showed that DSRX performance was similar to DSRCEDAR published results, but in some cases up to 5% worse. We attribute this difference to the fact that, unlike DSRX, DSRCEDAR modifies the MAC layer control messages and uses promiscuous mode to improve its core broadcast efficiency.

We would also like to have compared our approach to flat, hybrid protocols such as ZRP. In fact, we did simulate ZRP, but found it performed extremely poorly compared to the other algorithms for our workload and for all ZRP parameter settings. For example, with a zone-radius of 2, the best possible setting in our simulations, ZRP provided only 22% packet-delivery ratio for the simple case of a single, long-lived connection in an environment of 200 nodes and 5 m/s, maxiumum-speed mobility. We suspect the problem may be with the Glomosim implementation of ZRP available from the author's web page, but implemented by others, a suspicion the authors shared in private communication. As a result, we have excluded ZRP results from our analysis. While this omission is unfortunately, we believe that a direct comparison to ZRP would likely add little to the understanding of DCDS, because ZRP's inter-zone routing protocol is reactive and our environment does nothing to favour close-by communication, the situation where it does especially well. We would thus expect ZRP performance to be similar to that of DSR, which performs poorly compared to the hiearchical alternatives in our environment.

Finally, Table I lists the main DCDS parameter settings used in our simulations. We will further discuss heartbeat frequency and in Section IV-H.

B. Varying the Number of Hot Spots

Figure 2 shows six graphs that compare the protocols and vary the number of hot spots between 10 to 60 (i.e., 5% to 30% of the network); fixing the number of concurrent connections at 40, the lifetime of each connection at 10 seconds and mobility at 1–5 m/s. Each graph shows a different metric.

Figures 2a and 2c show packet delivery ratio and mean packet delivery latency, respectively. Figures 2b and 2d measure protocol overhead in different ways. Figure 2b provides the standard metric: the total number of control packets generated by the protocol. Figure 2d, shows a more accurate gage of energy consumption: the total number of packets, including data packets, *sent* in the MAC layer. Not shown is the total number of packets *received*, which is a nearly identical graph, but where values are scaled up by a factor of 28. Finally, Figures 2e and 2f list the number of packets dropped in the MAC and IP layers, respectively; the IP layer drops packets when its send queue overflows.

DCDS outperforms the other algorithms in virtually every situation. The only exception is the case of 10 hot spots, where DCDS performs about the same BDSR and better than the others. The performance gap between DCDS and the other algorithms widens as the number of hot spots increases, because with more hot spots more unique routes are used and thus route-caching is less effective. DCDS prefetches most of these routes into local caches before the routes are needed, but the other protocols discover the routes on demand and thus suffer from significantly higher network congestion. With 60 hot spots, for example, DCDS delivers 96% of packets successfully, DSRX only 53%, BDSR 32% and SDSR 17%. In this same situation, DCDS also has 12–45 times lower per-packet latency and 2.5–15 times lower message overhead. DCDS's lower latency illustrates the fact that few packet sends are slowed by the need to discover a route to the target node before sending the packet.

For the main performance metrics: delivery ratio, latency and overhead, the other hierarchical algorithm DSRX is consistently in second place for 40 or more hot spots, showing the advantage of cheaper route-discovery made possible by the backbone. When there are fewer than 40 hot spots, however, BDSR tends to perform somewhat better than DSRX. The reason for this difference is that when there are few hot spots, BDSR route-discovery requests are usually satisfied by nearby nodes thus limiting the extent of the discovery flood better than the DSRX backbone does. With 10 hot spots, for example, DSRX sends roughly ten times as many route-discovery messages as BDSR.

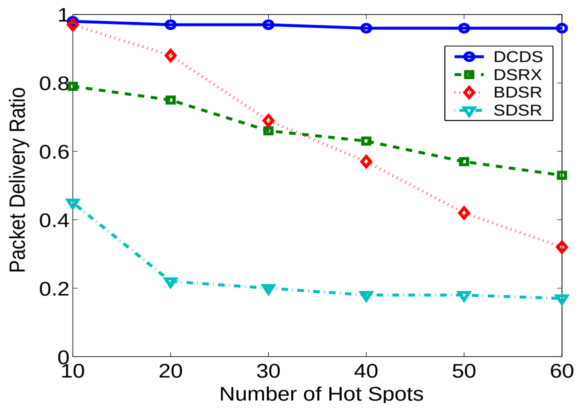
C. Detailed Evaluation of DCDS

Figure 3 shows how DCDS's performance gain can be attributed to each of the three novel features of the protocol. It reports packet delivery ratio and protocol overhead for seven versions of DCDS under the same conditions as in the previous section; the line labeled DCDS is the same as the corresponding line in Figure 2. For the other lines, the label indicates which of the three features is enabled. If a feature is not listed it is disabled; DCDS_BASE excludes all three features. The labels for the features are P for piggybacked prefetching, T for timestamped caches and R for proactive route adaptation and recovery.

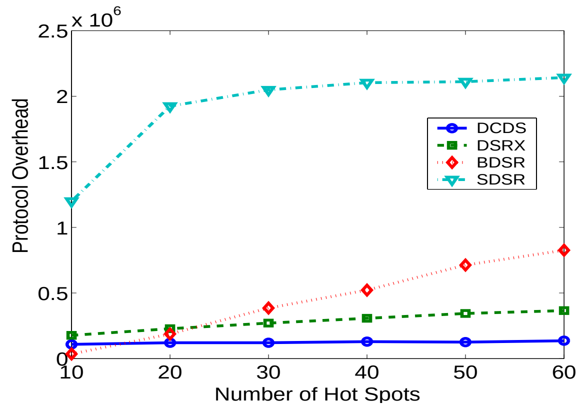
When each of the three features is examined in isolation, timestamped caches provide the biggest PDR benefit until the number of hot spots reaches 60, when prefetching edges ahead. This benefit is more than doubled, however, when the two features are combined. This behaviour is not surprising, as the features are largely complimentary. Prefetching puts more routes in caches and timestamping flushes stale routes from them. Prefetching without timestamps is only moderately effective, because getting the most up-to-date route into a cache isn't useful unless doing so discards other, obsolete routes from that cache. Similarly, timestamps by themselves can only improve performance if a cache stores multiple routes to a target from which to choose.

In terms of protocol overhead, prefetching alone delivers most of the benefit. The overhead that prefetching eliminates are route-discovery messages, a significant cost. With timestamping alone, the protocol is able to choose the best cached route when it has alternatives, but route-discovery is still required for any route that is not cached.

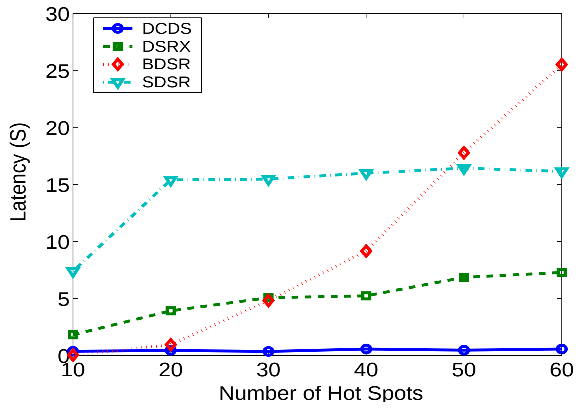
Finally, we see that proactive route adaptation and recovery significantly increases PDR when added to the other two features, but has little effect on overhead. This behaviour shows that proactive, local route information allows DCDS to re-route many packets that would otherwise be dropped, but with little additional overhead. This improvement is similar to what one would hope



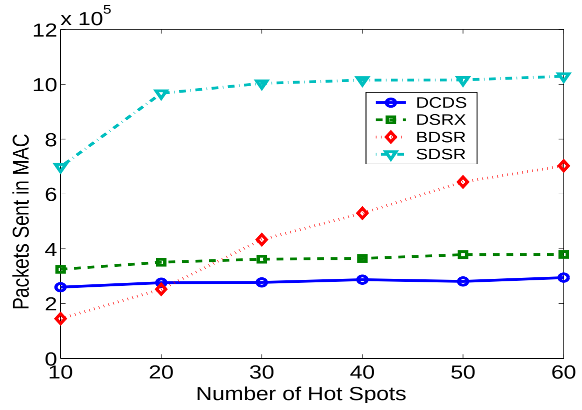
(a) Packet Delivery Ratio



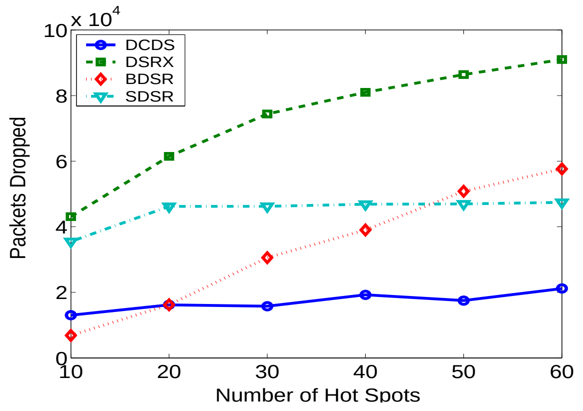
(b) Protocol Overhead



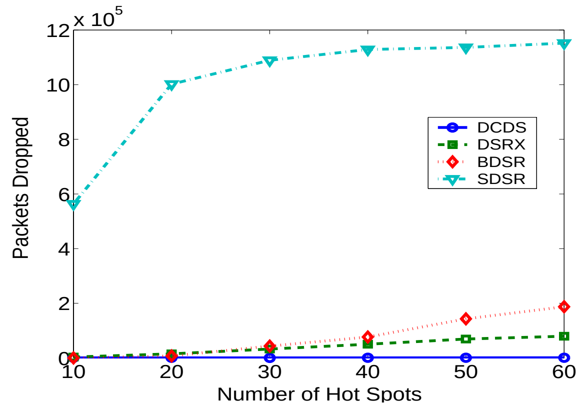
(c) Latency



(d) Packets Sent in MAC Layer



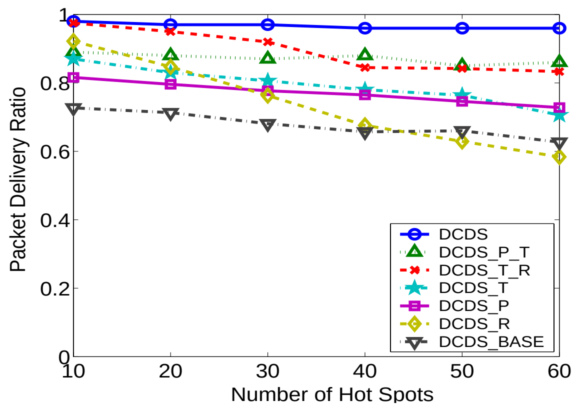
(e) Packets Dropped in MAC Layer



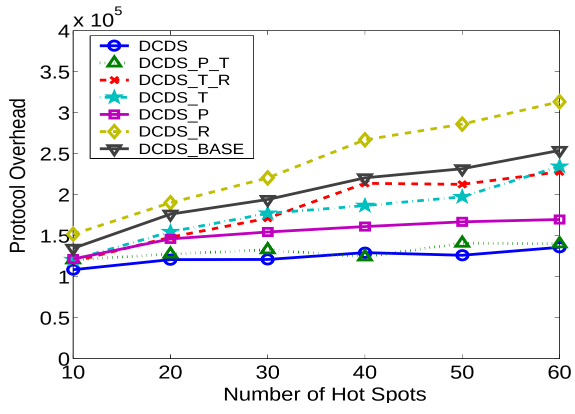
(f) Packets Dropped in IP Layer

Fig. 1. Scalability by Varying the Number of Hot Spots

to see from the packet salvaging schemes in other protocols. The danger with any such scheme is that when delivery fails due to congestion, as opposed to mobility or failure, salvaging can increase overhead without improving PDR. We see this effect with DCDS when proactive recovery is used without the other two features and the number of hot spots is greater than 40. In this case, the large number of route-discovery messages causes significant network congestion and packet salvaging only makes things worse.

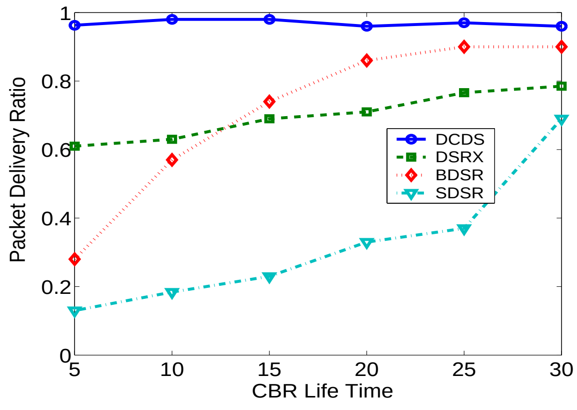


(a) Packet Delivery Ratio

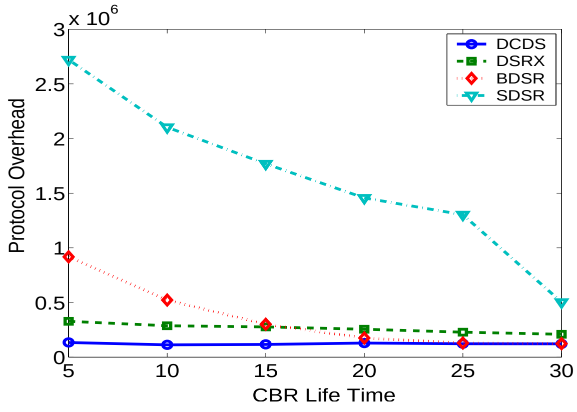


(b) Protocol Overhead

Fig. 2. Gain from Individual Design Choice



(a) Packet Delivery Ratio



(b) Protocol Overhead

Fig. 3. Increasing Connection Lifetime

D. *Increasing Connection Lifetime*

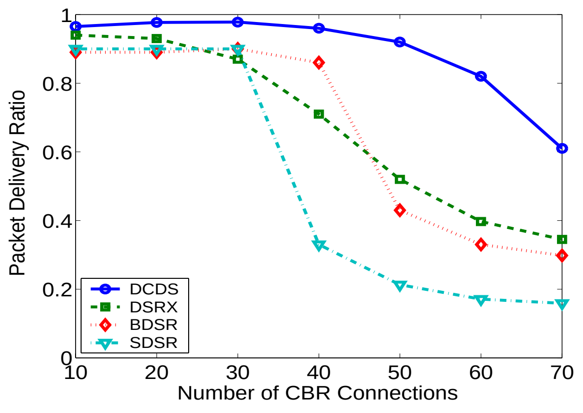
We have shown that DCDS substantially outperforms other approaches when new routes must be discovered frequently; for example, connections are short lived. Published results for other protocols typically examine only long-lived connections, an environment in which route discovery is rare. Figure 4 compares DCDS to the other algorithms as we increase the duration of each connection and thus require fewer new routes to be discovered. We vary connection lifetime from 5 s to 30 s; our previous results were for 10 s lifetimes. The number of hot spots is fixed at 40. The other parameters are the same as in the previous sections: 40 concurrent connections, 1 packet/s per-connect send rate, and 1–5 m/s mobility.

The results indicate that, while DCDS's performance improvement narrows as expected, even at 30 s, it maintains a 96% to 91% delivery-ratio lead over the next-best algorithm, BDSR. The best that DSRX, the other hierarchical protocol, does is 79%.

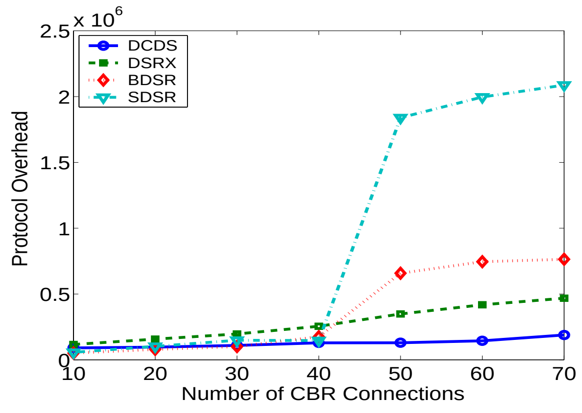
Not shown in the figure is what happens beyond 30 s. Our data show that for connections of 50 s, BDSR performs about as well as DCDS at 93%; DSRX is 83% and SDSR is 90%. If connections last the entire 850 s of the simulation, both BDSR, 96%, and SDSR, 94%, outperform DCDS at 91%; DSRX is still worse at 86%. However, if as few as 10% of the connections are short-term (i.e., 10 s), DCDS's performance improves to 96.5%.

Two factors explain DCDS's degraded performance when all of the connections last 850 s. First, after the first few seconds there are no new route discoveries at all, other than those caused by the use of invalid routes, so there are few opportunities to piggyback route changes that occur due to mobility. Second, DCDS's packet salvaging scheme is less aggressive than DSR's and thus the DSR variants are able to recover from more errors than we can.

Perhaps a more realistic scenario is one where some connections are short-term and others last longer. DCDS out performs the others even if a minority of connections are short-lived. For example, if 25% of the connections are short-lived, at 10 s per connection, and the remainder long-lived, at 850 s per connection, DCDS delivers 97% of packets, BDSR 90%, SDSR 64% and DSRX 72%.

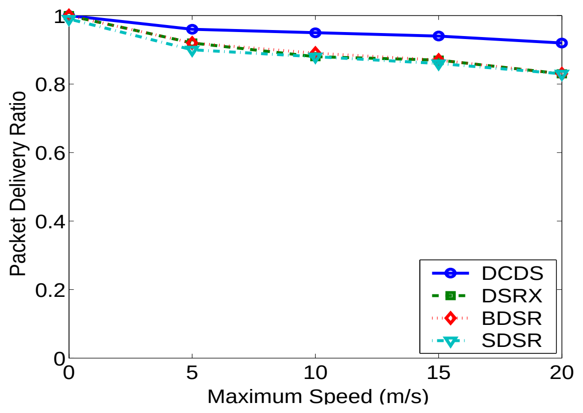


(a) Packet Delivery Ratio

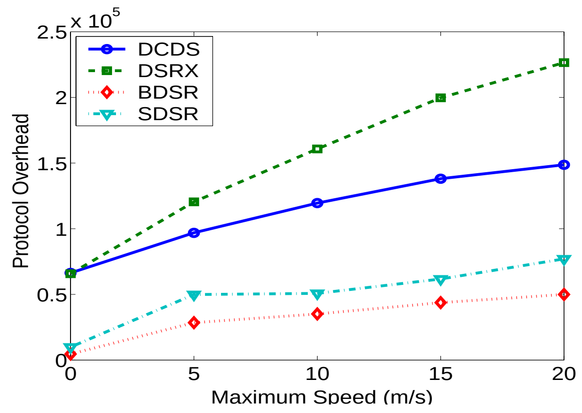


(b) Protocol Overhead

Fig. 4. Increasing the Number of Concurrent Connections



(a) Packet Delivery Ratio



(b) Protocol Overhead

Fig. 5. Increasing Mobility

E. Increasing the Number of Connections

Figure 5 compares the algorithms as we vary the total number of concurrent connections between 10 and 70. Increasing the number of connections increases overall network load and thus the graphs in this figure show how each algorithm deals with increasing network congestion. The other parameters are: 40 hot spots, 20-s connection lifetime, 1 packet/s per-connection send rate and 1–5 m/s mobility.

We see that as network congestion increases, all of the protocols, including DCDS have increasing difficulty delivering packets successfully, as we would expect. DCDS maintains its substantial lead over the other algorithms, delivering 61% of packets compared to second-best DSRX's 35% when there are 70 concurrent connections. Even in this case, however, DCDC incurs little additional protocol overhead.

F. Increasing Node Mobility

Figure 6 compares the algorithms when node mobility varies from no-mobility to mobility chosen randomly between 1 and 20 m/s. The other parameters are: 10 hot spots, 10 concurrent connections and 1 packet/s, per-connection send rate. The values of these parameters are less than in previous experiments in order to isolate the effects of mobility from those of congestion. At these modest settings none of the four algorithms experiences significant congestion. DCDS improvement over the other algorithms increases as congestion increases, as we have already shown.

The graphs show that, even in the absence of significant network congestion, DCDS achieves the best packet delivery ratio as mobility increases. At 1–20 m/s, for example, DCDS delivers 92% of packets, while the best of the others delivers 83%.

On the other hand, DCDS and DSRX, the two hierarchical protocols, have higher overhead than the two DSR variants due to the fixed cost of their proactive components and the cost of the additional route discoveries induced by mobility. In DCDS, for example, overhead peaks at 16 control packets per data packet.

When high mobility is combined with network contention, DCDS dominates the other algorithms by a substantial margin. In a configuration of 40 hot spots, 20 concurrent connections and 1–20 m/s mobility, DCDS successfully delivered 85% of its packets, BDSR 66%, SDSR 74% and DSRX 43%. These results also show

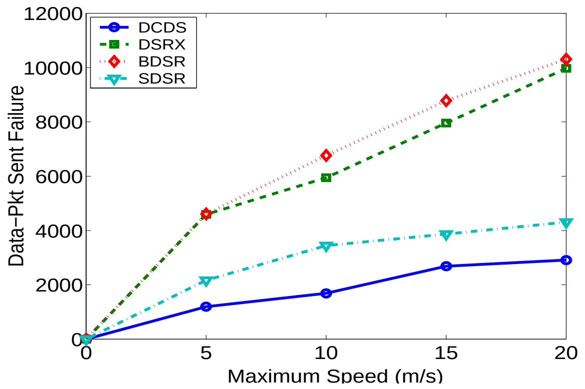


Fig. 6. Effects of Stale Routing Caches

TABLE II
PIGGYBACKING SIZE EFFECTS

Piggyback Limit	PDR	Protocol Overhead	Latency
10	61%	308,699	4.58 s
20	78%	222,801	2.39 s
30	80%	219,143	2.00 s
40	82%	217,694	1.94 s
50	81%	216,534	1.83 s
60	83%	208,791	1.72 s
no limit (69)	85%	200,258	1.59 s

that BDSR's large cache can sometime harm its performance compared to SDSR.

The main problem with mobile nodes is that they can invalidate cached routes in which they play a role. Figure 7 isolates this impact by counting the number of times each algorithm attempted to use a broken link between two nodes. In this experiment, the fact there is no congestion means that links break only when one of the link endpoints moves. This graph thus measures the effect of keeping stale routes in caches. We see that when mobility is high, both BDSR and DSRX are impacted by stale routes much more than the other DCDS or SDSR. DCDS does well, because the combination of timestamps and prefetching keep caches more up-to-date than the other algorithms. The reason that SDSR is better is that its caches are smaller than BDSR. High-mobility is a case where big caches do not improve the performance of DSR.

G. Scalability of Piggybacked Prefetching

A significant feature of our algorithm is its ability to piggyback multiple dominatee-dominator pairings in a single route-discovery reply message. The effectiveness of this technique relies on the assumption that the margin cost increasing message size is small compared to the benefit of more having more routes cached. When a message reaches its maximum size, however, the marginal cost of additional piggybacking, which would require an additional message, goes up substantially.

Two parameters that we have thus far held constant can increase the amount of piggybacked data generated by our algorithm. First, if node density increases, the number of domintees assigned to each domiantor will also increase and thus each dominator will seek to piggybacking more pairings. Second, if the network area increases, the lengths of network paths will also increase and thus more dominators will seek to add pairings

TABLE III
HB FREQUENCY EFFECTS WITH LOW MOBILITY

HB Intervals (DOT, DOE)	PDR	Protocol Overhead	Latency
(50 ms, 500 ms)	5%	2,312,548	13.94 s
(100 ms, 1000 ms)	28%	1,239,908	5.62 s
(250 ms, 2500 ms)	95%	171,867	0.65 s
(500 ms, 5000 ms)	92%	129,131	0.90 s
(1000 ms, 10000 ms)	88%	116,247	1.35 s
(2000 ms, 20000 ms)	76%	148,090	2.54 s

TABLE IV
HB FREQUENCY EFFECTS WITH HIGH MOBILITY

HB Intervals (DOT, DOE)	PDR	Protocol Overhead	Latency
(50 ms, 500 ms)	8%	1,963,431	11.35 s
(100 ms, 1000 ms)	53%	733,004	3.25 s
(250 ms, 2500 ms)	89%	247,386	1.34 s
(500 ms, 5000 ms)	84%	196,351	1.53 s
(1000 ms, 10000 ms)	75%	191,435	2.43 s
(2000 ms, 20000 ms)	61%	209,081	4.09 s

to each route-discovery reply message. A sufficient increase in either parameter cause reply messages to reach their maximum size before all pairings that could otherwise be piggybacked are added to the message.

Table II evaluates the limits of piggybacking scalability by simulating a high-mobility scenario and artificially constrained the number of dominator-dominatee pairings that can be piggybacked. We use the same settings as the high-contention, high-mobility case discussed in Section 4.6: 40 hot spots, 20 concurrent connections, 1 packet/s, per connection send rate and 1–20 m/s mobility. These values represent a case where piggybacking provides the biggest benefit: high mobility means that routes are invalidated frequently and thus must be refreshed frequently; 40 hot spots means that there is enough locality that prefetched routes are often used, so much that prefetching is caching alone would do about as well.

Due to IP MTU limit, the maximum number of piggyback entries that if in a reply packet is 69. Table II shows what the PDR, protocol overhead and latency of DCDS would have been if the limit were set lower. The limit is express as the maximum number of pairings allowed. The protocol adds pairings to the reply packet until it reaches this limit and then stops.

This data suggests that a network that was either twice as dense or in which routes were twice as long might see PDR drop from 85% to around 81% and see overhead increase by about 5%. Section 4.6 gives the performance of the other algorithms in this scenario, the best of which, SDSR is 74%.

H. Backbone Heartbeat Frequency

Finally, we examine the sensitivity of our algorithm to changes in the frequency of the DOMINATOR and DOMINATEE messages it uses to proactively maintain inter-dominator connections. To do this we ran two sets of simulations fixing all parameter settings and varying the heartbeat frequencies. The first, reported in Table III, uses the same settings as Section IV-E, fixing the number of concurrent connections at 50. The second, reported in Table IV, uses the same settings as Section IV-F, fixing mobility at 1–20 m/s.

In both cases, DCDS achieves the best packet delivery ratio and latency when the DOT interval is set to 0.25 s and the DOE interval is set to 2.5 s. From that point, its performance deteriorates with both increasing or decreasing frequency. The tradeoff is that if heartbeats are too frequent they create congestion and if they are to infrequent the longer it takes the backbone to adapt to backbone disconnections caused by node failure or mobility. The actual settings for these parameters used in the rest of the paper are slightly larger than optimal (i.e., 0.5 s and 5 s) to avoid biasing the results with values carefully tuned to characteristics of this particular workload. If we had used the optimal settings our results for DCDS would have been slightly better than we actually report.

V. Conclusion

Transient communication posses a difficult challenge to existing multi-hop, mobile, ad-hoc routing algorithms. The problem is that this environment requires more frequent route discovery than environments where connections are long lived. In existing protocols, route discovery is expensive and if it is performed too frequently it can bring the protocol to its knees and leave the network hopelessly congested.

This paper describes the design of a backbone-based, hybrid routing protocol called DCDS that works well in this environment. The protocol has three novel features. First, route information is prefetched into caches by piggybacking multiple routes in a single route-discovery reply message. Second, cache entries are timestamped, providing a heuristic for discarding redundant, out-of-date routes from caches. Third, cached routing information specifies only the backbone nodes on the path to the target, leaving each backbone node free to select any two- or three-hop path to the next backbone node on a route. This routing step occurs on demand using neighbourhood connectivity information that backbone nodes maintain proactively.

We use Glomosim to simulate our algorithm, two versions of DSR and a model backbone algorithm we implemented called DSRX. Our algorithm significantly out performs the others when connections are short, the network congested or mobility high.

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

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

[5] A. Iwata, C.-C. Chiang, G. Pei, M. Gerla, and T.-W. Chen, "Scalable routing strategies for ad hoc wireless networks," *IEEE Journal on Selected Areas in Communications*, pp. 1369–1379, Aug. 1999.

[6] S. Ramanathan and M. Steenstrup, "Hierarchically-organized, multihop mobile networks for multimedia support," *Mobile Networks and Applications*, vol. 3, no. 1, pp. 101–109, 1999.

[7] R. Sivakumar, P. Sinha, and V. Bharghavan, "Enhancing ad hoc routing with dynamic virtual infrastructures," in *Proceedings of IEEE Conference on Computer Communications (INFOCOM'99)*, Anchorage, AK, USA, Aug. 1999, pp. 1763–1772.

[8] M. Jiang, J. Li, and Y. C. Tay, "Cluster based routing protocol(cbrp)," Internet-Draft, draft-ietf-manet-cbrp-spec-01.txt, 1999, Work in progress.

[9] M. Balazinska and P. Castro, "Characterizing mobility and network usage in a corporate wireless local-area network," in *1st International Conference on Mobile Systems, Applications, and Services (MobiSys'03)*, San Francisco, CA, USA, May 2003.

[10] A. Balachandran, G. M. Voelker, P. Bahl, and P. V. Rangan, "Characterizing user behavior and network performance in a public wireless lan," in *Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems (SIGMETRICS'02)*, Marina Del Rey, CA, USA, 2002, pp. 195–205.

[11] T. Henderson, D. Kotz, and I. Abyzov, "The changing usage of a mature campus-wide wireless network," in *Proceedings of the 10th annual international conference on Mobile computing and networking (MobiCom'04)*, Philadelphia, PA, USA, 2004, pp. 187–201.

[12] M. McNett and G. M. Voelker, "Access and mobility of wireless pda users," in *Technical Report CS2004-0728, Department of Computer Science and Engineering, University of California, San Diego*, Feb. 2004.

[13] D. J. Baker and A. Ephremides, *The architectural organization of a mobile radio network via a distributed algorithm*. IEEE Transactions on Communications, COM-29, Nov. 1981.

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

[15] L. Bao and J. J. Garcia-Luna-Aceves, "Topology management in ad hoc networks," in *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*. ACM Press, 2003, pp. 129–140.

[16] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris, "Span: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks," *Wireless Networks*, vol. 8, no. 5, pp. 481–494, 2002.

[17] C. A. Santivanez, R. Ramanathan, and I. Stavrakakis, "Making link-state routing scale for ad hoc networks," in *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*. ACM Press, 2001, pp. 22–32.

[18] 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.

[19] 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.

[20] R. Sivakumar, B. Das, and V. Bharghavan, "The clade vertebrata: Spines and routing in ad hoc networks," in *Proceedings of the International Symposium on Computer Communications(ISCC'98)*, 1998.

[21] K. Xu, X. Hong, and M. Gerla, "An ad hoc network with mobile backbones," in *Proceedings of IEEE International Conference on Communications (ICC 2002)*, New York City, Apr. 2002.

[22] K. Xu and M. Gerla, "A heterogeneous routing protocol based on a new stable clustering scheme," in *Proceedings of IEEE MILCOM 2002*, Anaheim, CA, Oct. 2002.

[23] P. Sinha, R. Sivakumar, and V. Bharghavan, "CEDAR: a core-extraction distributed ad hoc routing algorithm," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 8, pp. 1454–1465, Aug. 1999.

[24] 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.

[25] 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.

[26] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," in *Proccedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*, Boston, MA, USA, Dec. 2002.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] CMU Monarch Group, "The CMU Monarch Project's Wireless and Mobility Extensions to NS," Aug. 1998.

[32] 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.