Routing Transient Traffic in Mobile Ad Hoc Networks

Kan Cai
Department of Computer Science
University of British Columbia
Vancouver, B.C., Canada
kcai@cs.ubc.ca

Michael J. Feeley
Department of Computer Science
University of British Columbia
Vancouver, B.C., Canada
feeley@cs.ubc.ca

Norman C. Hutchinson
Department of Computer Science
University of British Columbia
Vancouver, B.C., Canada
norm@cs.ubc.ca

ABSTRACT

Recent research shows that the traffic in public wireless networks is mostly transient and bursty. There is good reason to believe that ad-hoc traffic will follow the same pattern as its popularity grows. Unfortunately transient traffic generates route discoveries much more frequently than the well-studied long-term, constant-bit-rate traffic, causing network congestion problems for existing routing protocols. This paper describes the design of a new routing algorithm, called ECBR, that uses hybrid backbone routing in a manner that is well suited to workloads that include transient traffic. Our simulation results show that ECBR outperforms one of the main reactive algorithms (i.e., DSR). We also explain three key features of our algorithm and demonstrate their roles in substantially improving the performance compared to existing backbone routing techniques.

1. INTRODUCTION

A variety of ad-hoc routing protocols have been proposed and evaluated over the past several years. These include reactive protocols such as DSR [1] and AODV [2], proactive non-hierarchical protocols such as DSDV [3], hybrid non-hierarchical protocols such as ZRP [4] and SHARP [5], and hierarchical backbone protocols such as such as HSR [6], HierLS [7], DSRCEDAR [8] and CBRP [9].

It is standard practice to evaluate these algorithms using long-term, constant-bit-rate traffic [4, 5, 8, 10–12]. A key characteristic of this workload is that route discovery is infrequent with its cost amortized over a large number of data packets sent using each route. If some connections are more transient or traffic is more bursty, however, route discovery becomes more frequent and the performance of existing algorithms degrades to an extent not anticipated by previous constant-bit-rate studies.

In fact, there is good reason to believe that real ad-hoc networks will indeed include transient and bursty traffic. Previous analyses of public wireless networks in both academic and corporate environments [13, 14], for example, have shown that users are often passive and network traffic is bursty. In these studies, wireless sessions were usually short-lived and the long-term connections that did exist were idle much of the time. Similar studies of PDA users [15, 16] have shown that median user-session duration is 5-6 minutes and that users switch access points every 1.8 minutes. Furthermore, applications such as web browsers and instant messaging are inherently bursty. Message bursts from these applications are typically as short as a few seconds and are often separated by long periods of inactivity.

We show in this paper that backbone approaches are better suited to transient traffic than flat, reactive algorithms such as DSR. The reason is that when route discovery is frequent, reactive algorithms flood the network with too many route-discovery messages, triggering broadcast storms that render the entire network temporarily useless. Backbone algorithms, on the other hand, confine routing to the backbone, which is comprised of only a small subset of the network. As a result, fewer messages are required to discover new routes, because route caching is more effective and route-discovery broadcasts are forwarded by fewer nodes.

This paper describes a hybrid, backbone routing algorithm we have developed called ECBR (end-point cache backbone routing). ECBR is similar to other backbone protocols such as DSRCEDAR and CBRP, but with three key differences related to how it acquires and caches routing information. First, ECBR piggybacks additional information on each route-discovery reply message to prefetch routes for multiple nodes. Second, ECBR uses timestamps on route-cache entries to keep cached routing information current. Third, ECBR uses proactive, local route adaptation and recovery to dynamically change inter-backbone-node links without changing globally cached routes or dropping packets.

Our evaluation shows that ECBR can robustly handle a wide range of traffic patterns, including both short-term and long-term connections. For transient traffic, ECBR significantly outperforms DSR and a DSRCEDAR-like algorithm we implemented for comparison purposes, called DSRX.

2. BACKGROUND AND RELATED WORK

Research on clustering and backbone routing in mobile ad hoc networks has a long history, originating from radio networks [17, 18]. 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.

2.1 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. 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, however, 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 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.

The main drawbacks of backbone routing are unfairness, route inefficiency and backbone bottlenecks. These problems are the price paid for the efficiencies the backbone delivers in other ways, primarily reducing congestion caused by route discovery. Since much of the network load is focused on nodes in the backbone, a node pays a high price if it is selected to be part of backbone. It is therefore desirable to rotate the role of cluster head among the nodes in the network [19, 20]. Backbone routing also uses sub-optimal routes, requiring more hops than necessary [21]. While we haven't done so, it would be straightforward to add features such as these to ECBR.

Finally, the backbone algorithms share a fundamental drawback: the backbone can become a bottleneck that limits aggregate network bandwidth. The backbone, being the only routing path in the network, has to sustain all the data and control packets. In other protocols these packets could be forwarded using non-interfering routes.

There are two *complementary* methods to alleviate the bottleneck's impact. One is to reduce the backbone complexity by building a small connected graph in the network. Building a backbone that has good approximation ratio to the ideal Minimum Connected Dominating Set (MCDS) has been the research focus of ad hoc clustering in recent years [22–28]. Another method is to reduce the protocol overhead forwarded on the backbone. This can greatly help the backbone to deal with transient traffic since, given a fixed network load, the major issue imposed on the backbone by transient traffic is the increasing number of route discoveries. ECBR follows the second method.

2.2 Related Work

Multi-hop, ad-hoc, wireless protocols can be roughly be divided into two categories, *flat* or *hierarchical*, based on the way they organize routing information. 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 [3] 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 [1] and AODV [2], on the other hand, discover and maintain a route only when needed and thus have no fixed overhead and can adapt quickly to topology changes. But 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 [6] and HierLS [7], for example, build multi-level, multi-hop clusters and then proactively maintain global routes among the clusters. Other *hierarchical* approaches such as DSRCEDAR [8] and CBRP [9] proactively maintain a spanning backbone and then use the backbone to perform route-discovery broadcasts reactively and more efficiently.

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 on top of the backbone in a more modular fashion.

Hierarchical link-state protocols such as HSR [6] and HierLS [7] follow the more integrated approach. They build multi-level and multi-hop clusters, and then proactively maintain routes 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. [27, 29]. Compared to HSR and HierLS, the spine approach is simpler, because it uses only two layers in the routing hierarchy. One of the spine papers [29] provides a formal description of an algorithm that uses a local-recovery scheme similar to ours, but does not go beyond this formal description. Also due to the formal nature of this work, they assume that the radio network provides a reliable broadcast scheme, 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 ECBR. There are four other algorithms that also use this layered scheme.

First, K. Xu et al. [30, 31] 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. [9] describe a DSR-like routing algorithm on top of a backbone, called CBRP. All nodes send periodic, one-hop broadcast messages, which 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. 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. [32] propose a core-based routing algorithm, called CEDAR, which includes various features designed to support QoS routing. Each core node establishes tunnels with its neighbouring core nodes via three-hop broadcast messages. 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.

Fourth, P. Sinha et al. subsequently improved CEDAR by reducing its core broadcast overhead [8]. 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. Their results show that the addition of the backbone improves the performance of both of these standard, flat algorithms.

2.3 Discussion of ECBR's Uniqueness

In many ways our work builds on earlier systems such as CBRP and DSRCEDAR that layer DSR or a similar algorithm on top of a backbone. Our contribution is to extend the basic, reactive routing algorithm to better handle transient communication and the inherent bottleneck drawback of backbones. What makes ECBR unique 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. Third, it uses proactively maintained neighbourhood information to adapt to changes in inter-backbone-node connectivity without invalidating globally cached routes or dropping packets. Note that these features can be adopted by any backbone regardless of variant backbone-construction algorithms.

The three features of ECBR are inspired by similar ideas in previous flat algorithms such as DSR and AODV. Piggyback prefetching is inspired by DSR. DSR is designed to use *source routing* and thus inherently also piggybacks extra routing information in all packets that it generates. In fact whenever a DSR node receives or overhears a packet in the network, it learns not only the routes to reach the source and destination but also to all the nodes on that path. Similar to DSR source routing, an extension of AODV, AODVbis [33], also uses a scheme called *path accumulation* to disseminate aggregated routing information during route discovery.

However, ECBR piggybacks topology information even more aggressively than both DSR and AODVbis. In ECBR, each cluster head proactively maintains all the last-hop topology information to the nodes in its cluster. Therefore, in addition to using source routing on the backbone, ECBR can also piggyback all the last-hop topologies in a route reply packet until the packet size reaches the IP MTU limit. In our ECBR implementation, the maximum number of last-hop entries that a route reply message can include is 69. This is the main reason that enables ECBR to substantially reduce its protocol overhead.

Cache timestamps play a similar role to sequence numbers in AODV (also DSDV) in that both are used to indicate route freshness. The difference between timestamps and AODV sequence numbers is that timestamps are stored on only the backbone nodes and require no additional messages to be propagated. When propagating, these timestamps are only spead to the other nodes in the backbone instead of the rest of the network.

Finally, ECBR tries to locally recover broken backbone links before turning to DSR's packet salvage scheme. This local recovery is inspired by algorithms such as ABR [34] and ADMR [12] to limit the impact of global route repairs. However, unlike these protocols using TTL-limited broadcasting messages to reactively locate the next hop or the destination when a link break happens, ECBR uses an *incomplete* backbone path for routing and heartbeat messages to proactively maintain multiple 2- or 3-hop connections between two neighbouring cluster heads, and thus is able to fix most broken backbone links with no delay and little overhead.

The fundamental difference between ECBR's features and those in all the above mentioned algorithms is that ECBR is a backbone approach, while the others are flat algorithms. At cost of a fixed, low proactive overhead, the backbone scheme not only reduces the impact of global route discoveries but also makes these three features more effective; simply using piggybacking, timestamping and local recovery in flat algorithms will not achieve as much performance gain as ECBR. ECBR aggregates routing information in the backbone and thus allows a cluster head to learn and piggyback a large number of routes in a single route reply message. Timestamps are also aggregated in the dominators and disseminated as part of piggybacking information. This permits dominators to be aware of topology changes with little overhead. The backbone construction and maintenance naturely enables two-layer routing and local route recovery. The next section presents ECBR in detail.

3. ECBR ALGORITHM

ECBR is partly proactive and partly reactive. Its proactive component acts to maintain a single backbone for the network. Routing, on the other hand, is performed reactively in a manner similar to DSR, but where routes are confined to follow the 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 entire network and no two dominators are in range of each other. The graph links dominators together using two- and three-hop links.

ECBR 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 and the other to plan a backbone route to that dominator. Both of these are maintained *reactively*.

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. ECBR 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 connections between a pair of dominators are lost.

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

3.1 Proactive Backbone Construction

The basic operation of the maintenance protocol is to group nodes into clusters around dominators. 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. Every node starts as a dominator until it discovers another dominator in its radio range that has a lower ID.

Each dominator periodically broadcasts a one-hop DOMINATOR heartbeat message that is timestamped and contains its node 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 such a heartbeat, it records the dominator in its local list and timestamps that entry with that dominator's timestamp.

Each dominatee periodically broadcasts a one-hop DOMINATEE heartbeat message that 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 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

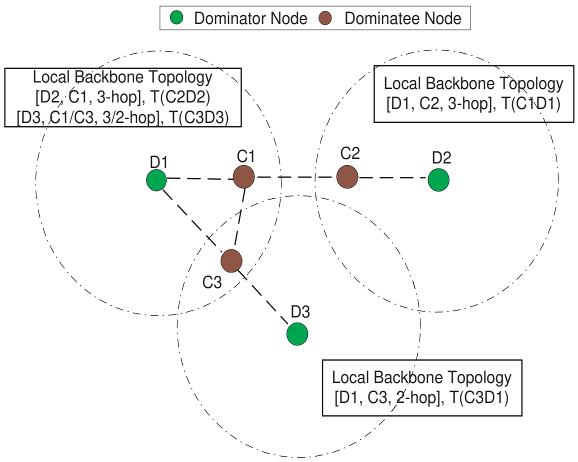


Figure 1: Example of a simple backbone.

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 connectivity 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 if 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, especially 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.

3.1.1 An example

Figure 1 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 in the connectivity list at each dominator once the protocol has reached steady state.

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 $C2$. Note that each backbone link is timestamped, and T(C2D2) stands for the last time that D2 sent a heartbeat message and C2 received it.

3.2 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 simply forwards 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, the upper routing layer 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 ECBR's caching and base routing scheme. Following this is a detailed discussion of the three key features of ECBR: cache timestamping, piggybacked cache prefetching and local route adaptation and recovery.

3.2.1 Routing Caches

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.

The DRT is updated by route-discovery reply messages, which typically include multiple dominatee-dominator pairings, due to the piggybacking feature described in Section 3.2.4. When a dominator receives or forwards a reply 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 backbone uses source routing for packet delivery.

3.2.2 Base Routing Scheme

ECBR 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 cannot 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.

One optimization of ECBR is that a newly-initiated route-discovery message is first propagated on the backbone following a *spanning tree* rooted at the requesting dominator, as an attempt to reduce the number of redundant route-discovery messages; the spanning tree is calculated locally using the backbone topology information in BRT. If this fails, the subsequent route-discovery retries use the above backbone broadcast scheme.

Whenever a dominator is unable to forward a packet to a neighbouring dominator, it performs the following error-recovery procedure. First, it deletes the corresponding backbone link 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.

3.2.3 Cache timestamping

A common problem with reactive caches is detecting routes that become invalid when nodes move or fail. The caches typically hold multiple paths to each target. Some are invalidated by a particular failure and some aren't, but it can be costly to determine which is which. Using an outdated route for packet delivery may cause packets to be dropped.

In ECBR 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 freshness of cached pairings and then using this freshness 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 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. Recall that these timestamps are orginated at the dominatees, similar to the sequence numbers used in AODV. There is thus no need to use any global time synchronization algorithm in ECBR.

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 packet's route accordingly before forwarding the packet toward this updated location.

Timestamps are also used by the lower-level routing mechanism that connects neighbouring dominators to each other. Often a pair of dominators have many 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.

3.2.4 Piggybacked Cache Prefetching

The second important optimization in ECBR is that route-reply messages are padded to piggyback route information for multiple targets 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 ECBR, the two-layer 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 in reply-packet size.

ECBR 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 dominator. 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-message'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 IP MTU limit, if it does. Each also extracts entries from the packet to add to its own DRT.

3.2.5 Route Adaptation and Recovery

The final key feature of ECBR 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. Because 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 connector with the most recent timestamp, otherwise it chooses the connector of the most recent three-hop link. 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-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 no connections is available 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.

4. SIMULATION SETTINGS

4.1 The Scenarios

Our simulations use Glomosim [35], a scalable simulator with accurate physical-layer and radio-propagation models. We set Glomosim parameters as follows: bandwidth is 2 Mb/s, radio frequency is 2.4 GHz and transmission range is 250 m.

Most of 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 [10, 11] to avoid the formation of chain-like backbones, which would tend to favour backbone approaches. 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 dominators (std. dev.: 1.5) with an average dominator degree of 7 two- or three-hop neighbouring dominators (std. dev.: 0.8). We also vary network area from 1500m x 250m to 1500m x 1500m and will discuss its impact in Section 5.5.2.

Table 1: ECBR 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

Each simulation lasts 910 seconds. We avoid 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 [1] 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 Sections 5.1.3 and 5.2.3, we examine ECBR performance for a variety of faster velocities.

We adopt a multi-destination, varying-duration CBR traffic pattern similar to that used in the SHARP paper [5]. 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 destinations are chosen randomly from the list of hot spots. The duration of each CBR connection is another parameter that we vary. When one ends, another is chosen to take its place. We are thus able to simulate a wireless environments where transient traffic exists. The size of each CBR packet is 256 bytes and packets are generated at a fixed rate of 1 pkt/sec. We vary network load by changing the number of concurrent CBR connections.

4.2 The Protocols

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

SDSR is the standard version of DSR. To ensure our comparison was faithful to previously reported DSR results, we imported the DSR code from *ns2* [36] and modified it to work with Glomosim. We verified that this implementation closely matches the *ns*2 version when physical values are set as suggested by [37].

BDSR is the same as SDSR in all respects except that each node uses a bigger routing cache: 200 entries instead of 64. This 200-entry routing cache is about three times the size of the routing caches in a dominator including BRT, DOT and DRT; dominatees have an even smaller cache. 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 compared to SDSR when nodes are very mobile, as we show in Section 5.1.3.

DSRX is a protocol we implemented as a model for a class of hierarchical protocols that are similar to DSRCEDAR. It uses the same backbone algorithm as ECBR, 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, i.e., piggyback prefetching, cache timestamping and local, proactive recovery. 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. However, the published results for DSRCEDAR are for longer-term CBRs than those we focus on in this paper (i.e., roughly 800s) [8]. For this type of connection, our results showed that DSRX performance was similar to DSRCEDAR published results (using the ns2 settings in Glomosim), but in some cases up to 5% worse. We attribute this difference to the fact that 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 a 22% packet-delivery ratio for the simple case of a single, long-lived connection in an environment of 200 nodes and 5 m/s. 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 unfortunate, we believe that a direct comparison to ZRP would likely add little to the understanding of ECBR, 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 hierarchical alternatives in our environment.

Finally, Table 1 lists the main ECBR parameter settings used in our simulations. Note that we fix the heartbeat intervals without adaptively tuning them in different scenarios to strike the balance between congestion and ECBR's responsiveness to topology changes. In fact, we did examine the sensitivity of ECBR to changes of heartbeat frequencies and found that the optimal setting for heartbeat intervals is 0.25 s for dominators and 2.5 s for dominatees in most of the cases we simulated. The combination of 0.5 s and 5 s ranks the second best with up to 5% worse performance in high-mobility scenarios. We omit the analysis here due to space limitation. This paper uses the more conservative settings, 0.5 s and 5 s, to avoid any biased results with values carefully tuned to characteristics of our particular workload.

5. EVALUATION

This section describes the simulation results. We first show that ECBR significantly outperforms DSR and DSRX in scenarios with multiple hot spots and transient traffic. We then provide detailed analyses of DSR's and DSRX's performance and break down ECBR to show how each of its features helps to improve performance. Finally, a discussion of ECBR's limitations is presented.

All of the data reported in this section is the mean of ten trials; the standard deviations are also presented. Note that we only show half of the symmetric deviation for each data point to avoid cluttering the figures.

5.1 The Impact of Transient Traffic

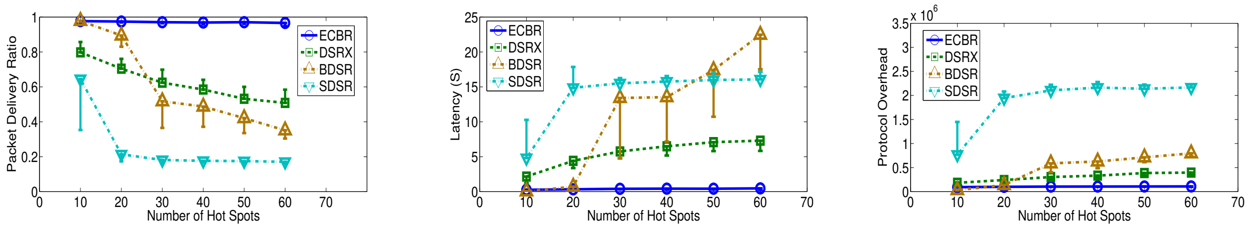
This section varies three parameters: the number of hot spots, CBR duration, and mobility, to examine the impact of transient traffic.

5.1.1 Varying the Number of Hot Spots

Figure 2 shows the impact of varying the number of hot spots from 10 to 60 (i.e., 5% to 30% of the nodes); the number of concurrent connections is fixed at 40, connection duration at 10 s and mobility at 1–5 m/s.

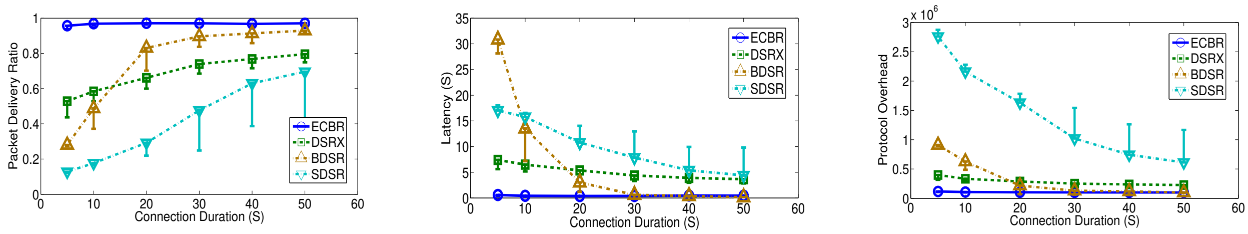
Figures 2(a), 2(b) and 2(c) show the packet delivery ratio (PDR), latency and protocol overhead, respectively. ECBR outperforms the other algorithms in virtually every situation; BDSR matches the performance of ECBR in the case of 10 hot spots. The performance gap between ECBR and the other algorithms widens as the number of hot spots increases.

Figure 2 also shows that ECBR performs the most con-



(a) Packet Delivery Ratio (b) Latency (c) Protocol Overhead

Figure 2: Increasing the Number of Hot Spots



(a) Packet Delivery Ratio (b) Latency (c) Protocol Overhead

Figure 3: Increasing Connection Duration

sistently among all these protocols: PDR of 97%, latency of 0.4 s and overhead of 10^5, with little deviation. The performance of other protocols, on the other hand, is affected by the randomnesses in the network including initial network topology, node movement and the source, destination pairs of CBR connections, etc. They usually have higher deviations especially when their performance starts to drop. The PDR deviations of BDSR and DSRX are around 10% and 7% when their performance declines, for example.

5.1.2 Increasing Connection Lifetime

Figure 3 shows the impact of increasing the duration of each connection. We vary connection lifetime from 5 s to 50 s. The number of hot spots is fixed at 40; the other parameters are the same as in Section 5.1.1.

We can see from Figure 3 that ECBR outperforms all the other protocols in every case. The results also indicate that, while ECBR's performance advantage narrows as expected, even at 50 s, it provides the best packet delivery ratio of 97%. Not shown in the figure is what happens beyond 50 s. Our data show that if connections last the entire 850 s of the simulation, both BDSR and ECBR perform 96%, and SDSR, 95%; DSRX is still worst at 85%. The deviations are small.

With a more realistic scenario in which network load consists of a mix of short-term and long-term connections, ECBR outperforms the others even if a minority of connections are short-lived. For example, if 25% of the connections are short-lived, at 5 s, and the remainder long-lived, at 850 s, ECBR delivers 97% of packets (std. dev.: 1.1%), BDSR 76% (std. dev.: 20%), SDSR 27% (std. dev.: 8%) and DSRX 68% (std. dev.: 5%).

5.1.3 Increasing Mobility

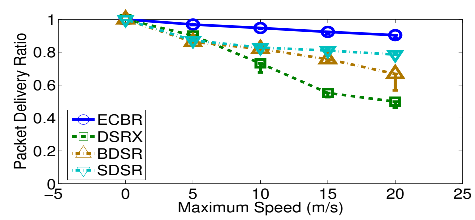


Figure 4: Increasing Mobility (20 CBRs, 40 Hotspots)

We evaluate how these protocols handle transient traffic under different mobility settings and show the results in Figure 4. We vary the maximum node mobility between 0 m/s (no mobility) and 20 m/s. The number of hot spots is fixed at 40, the number of concurrent connections at 20, CBR lifetime at 10 s.

ECBR again dominates the other algorithms by a substantial margin, which suggests that ECBR is best able to keep track of topology changes. In a configuration of 1–20 m/s mobility, ECBR successfully delivered 90% of its packets, BDSR 67%, SDSR 79% and DSRX 50%. These results also show that BDSR's large cache can sometime harm its performance compared to SDSR.

5.2 Analysis of DSR's Performance

The poor performance of DSR under transient traffic is largely caused by congestion. However, the stale caches also have an adverse impact on DSR's performance, especially in highly mobile environments.

5.2.1 Cache Misses

The fundamental reason that causes both BDSR and SDSR to perform poorly in Sections 5.1.1 and 5.1.2 is that transient connections cause more route-cache

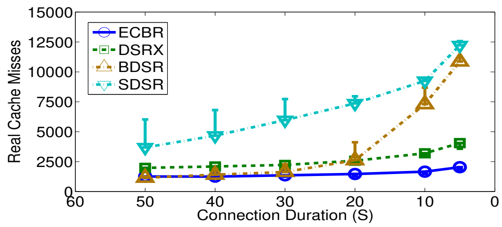


Figure 5: Cache Misses as CBR Duration Decreases

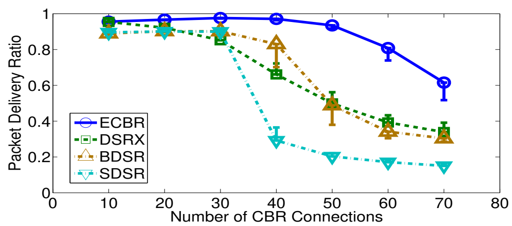


Figure 6: Increasing the Number of Concurrent CBRs

misses than long-term ones do, and thus generate more route-discovery messages that explains the substantial increases of protocol overhead shown in Figure 2(c) and 3(c), resulting in severe network congestion.

We count those *real* cache misses that indeed initiate network-wide route discoveries, and Figure 5 shows the result in the varying-CBR-duration scenario. We can see that the number of cache misses of both DSRs increases significantly as the connection duration decreases. This is because, given a fixed network load, reducing the CBR lifetime increases the number of connections in the network. Cache misses are thus more likely to happen for the DSR algorithms; even the 200-entry cache cannot sustain BDSR's cache hit ratio when the lifetime is reduced to 10 s although it helps BDSR to outperform SDSR [1].

Our results also show that an increase in hot spots causes both DSRs to experience more cache misses. For example, BDSR's cache misses jump from 2152 to 6832 as the number of hot spots increases from 20 to 30; SDSR's increases to 8610 when there are only 20 hot spots in the network. This is close to the maximum possible number of route discoveries that is bounded by the discovery timeout and discovery retry limit defined in DSR. This is because with more hot spots more unique routes are used and thus route-caching is less effective.

5.2.2 *The Number of Concurrent Connections*

Another factor that causes congestion is the traffic load. Figure 6 compares the algorithms as the total number of concurrent connections varies between 10 and 70, indicating how each algorithm deals with increasing network congestion. The other parameters are: 40 hot spots, 20-s connection lifetime and 1–5 m/s mobility.

We see that both DSRs can handle a small number of concurrent transient connections well. However, as network congestion increases, they have increasing difficulty delivering packets successfully since the 2 Mbps bandwidth can no longer handle the frequently-generated route discoveries. Also when the traffic load increases to 60 pkts/s, ECBR's backbone finally fails to sustain its performance due to the bottleneck issue.

5.2.3 *Mobility*

Transient communication can worsen the well-known stale-cache problem of DSR. This is because, with long-term connections, a source node uses a path continously and therefore can detect a broken link promptly. With transient connections, however, a source is likely to use a route for a short period of time and then cache it for later uses. This cached route might become inaccurate next time when the node uses it or provides it to other route discoveries due to topology changes, thus causing data packets to be dropped.

Figure 7 compares the algorithms when node mobility varies from no-mobility to mobility chosen randomly between 1 and 20 m/s. The other parameters are the same as in Section 5.1.3 except that we use only 10 hot spots and 10 concurrent CBRs to isolate the effects of mobility from those of congestion.

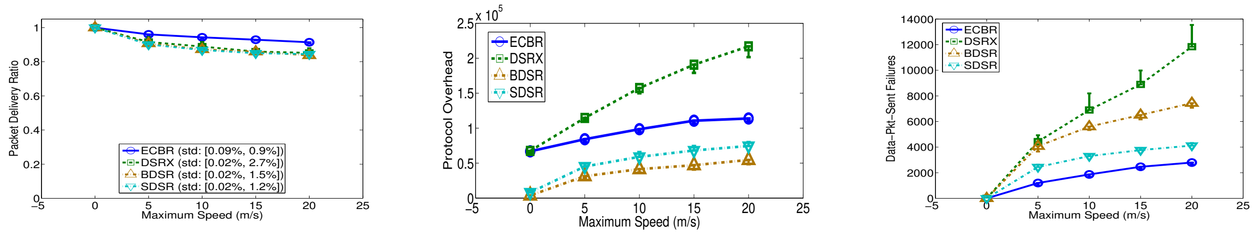
At these modest settings none of the four algorithms experience significant congestion; no packet is dropped due to IP queue overflow. In fact, Figure 7(b) shows that the two hierarchical protocols have higher overhead than the two DSR variants due to the fixed cost of their proactive backbone-maintenance component.

Figure 7(c) isolates this impact by counting the number of times each algorithm attempts 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. We see that when mobility is high, both BDSR and DSRX are impacted by stale routes much more than ECBR or SDSR. 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, as also shown in Figure 4.

5.3 Analysis of DSRX's Performance

We can see from Figures 2, 3 and 6 that even the naive hierarchical algorithm, DSRX, can consistently outperform the DSR variants in high-contention and low-mobility scenarios. This indicates that the backbone approaches are able to better handle transient traffic. However, DSRX's performance is always substantially worse than ECBR's, and in some cases it underperforms the DSRs as well. This section discusses the performance and limitations of DSRX in details.

[1]The reason why SDSR has less latency under congestion is because it can only deliver data packets via short paths in a highly saturated network.



(a) PDR (b) Protocol Overhead (c) Effects of Stale Caches

Figure 7: Increasing Mobility (10 CBRs, 10 Hotspots)

5.3.1 *Backbone is Better for Transient Traffic*

The backbone consists of only a small subset of the nodes in the network. Therefore, the cost of a single route discovery is likely to be less than the flat algorithms such as DSR. We can see from Figure 5 and Figure 3(c) that the ratio of standard DSR's protocol overhead to DSRX's is bigger than the ratio of their cache misses. Second, the backbone approaches aggregate the routing information in the cluster heads, and thus do not suffer as many cache misses as the DSR variants when the number of hot spots or unique connections increases as shown in Figure 5. Note that in less congested environments where the number of hot spots is small or the connection duration is long-lived, DSRX also tends to perform worse than BDSR. This is because, in these cases, BDSR route discoveries are usually satisfied by nearby nodes, thus limiting the extent of the flooding.

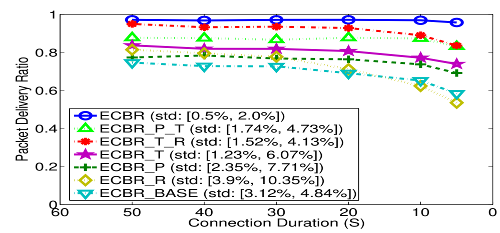
5.3.2 *The Bottleneck Backbone*

Even though the backbone is able to reduce the range and frequency of broadcast route discovery messages, it confines both data and control packets to the backbone, the only routing path in the network. This not only reduces the *aggregate bandwidth* (less spatial reuse), but also makes the backbone the potential bottleneck for performance improvement.

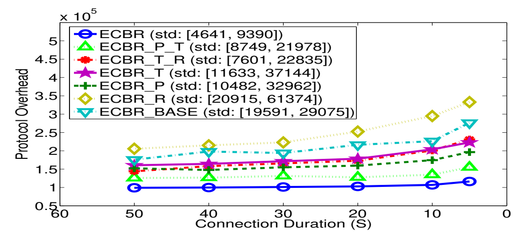
In the experiments where we vary the number of hot spots and CBR duration, DSRX generates at least twice as much overhead as ECBR. For example, the overhead of DSRX is about 3.6 or 3.4 times respectively when we use 60 hot spots or 5-second CBRs. This amount of control packets together with data packets has already caused enough congestion problem to break the backbone apart.

5.3.3 *Stale Caches*

We can also see from Figure 7(c) that DSRX suffers the most from stale caches. It simply lays DSR on top of the backbone and therefore drops as many data packets due to stale caches as BDSR at 5 m/s maximum mobility. It is worse than BDSR as node mobility increases because DSRX does not use the optimizations deployed in DSR such as gratuitous error messages and promiscuous mode to alleviate the impact of outdated routes. Note that these dropped packets result in more overhead that exacerbate the bottleneck problem for DSRX.



(a) PDR Gain



(b) Overhead Gain

Figure 8: Break-down of ECBR's Performance Gain

5.4 Detailed Evaluation of ECBR

ECBR outperforms both DSRs and DSRX by a substantial margin in all the scenarios, regardless of the degree of contention and mobility. This is because of the three unique features that it has adopted: piggybacked prefetching, cache timestamping, proactive route adaptation and recovery.

Figure 8 shows how ECBR's performance gain can be attributed to each of these three features. It evaluates seven versions of ECBR under the same conditions as Section 5.1.2; the line labeled ECBR is the same as the corresponding line in Figure 3. For the other lines, the label indicates which of the three features is enabled. If a feature is not listed it is disabled; ECBR_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. The min and max values of standard deviations are reported in the legend instead to avoid cluttering the figures.

When each of the three features is examined in isolation, timestamped caches provide the biggest PDR benefit, while prefetching alone reduces most of the overhead. The overhead that prefetching reduces are route-discovery messages. It results in more benefit, however,

Table 2: Piggybacking Size Effects

Piggyback Size	PDR (std.)	Overhead (std.)
10	79% (8.6%)	178,990 (24,141)
20	86% (5.1%)	157,840 (17,854)
30	88% (2.1%)	147,790 (10,484)
40	89.6% (2.3%)	146,250 (10,413)
50	89.0% (4.1%)	144,990 (14,114)
60	90.3% (2.7%)	140,540 (11,406)
No limit (69)	90.4% (1.7%)	140,330 (10,686)

when piggybacking and timestamping are combined. This behaviour is not surprising, as the features are largely complementary. 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 is not 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.

Finally, using proactive route adaptation and recovery alone always generates more protocol overhead because it keeps reporting route-error messages back to the upstreaming dominator whenever a backbone link breaks. However, when added to the other two features, we see that it not only significantly increases PDR but also reduces the overall overhead. This behaviour shows that proactive, local route information allows ECBR to reroute many packets that would otherwise be dropped and reduce the global route discoveries for recovery. The danger with any such scheme is that when delivery fails due to congestion not mobility or failure, salvaging can increase overhead without improving PDR. We see this effect with ECBR when proactive recovery is used without the other two features and the CBR duration is less than 10 seconds. In this case, the large number of route-discovery messages causes significant network congestion and packet salvaging only makes things worse.

5.5 Discussion of ECBR Limitations

5.5.1 Scalability of Piggybacked Prefetching

The effectiveness of piggybacked prefetching relies on the assumption that the marginal cost of increasing message size is small compared to the benefit of having more routes cached. When a message reaches its maximum size, however, additional piggybacking would require an extra message, which causes the marginal cost of the additional piggybacking to increase 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, 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 to each route-discovery reply message. A sufficient increase in either parameter will cause reply messages to reach their maximum size before all pairings that could otherwise be piggybacked are added to the message.

Table 2 evaluates the limits of piggybacking scalability by simulating a high-mobility, high-contention scenario while artificially constraining the number of dominator-dominatee pairings that can be piggybacked. We use the same settings as in Section 5.2.3. These settings 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.

Due to the IP MTU limit, the maximum number of piggyback entries that fit in a reply packet is 69. Table 2 shows what the PDR and protocol overhead of ECBR would have been if the limit were set lower. 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 90% to around 88% and see overhead increase by about 5%.

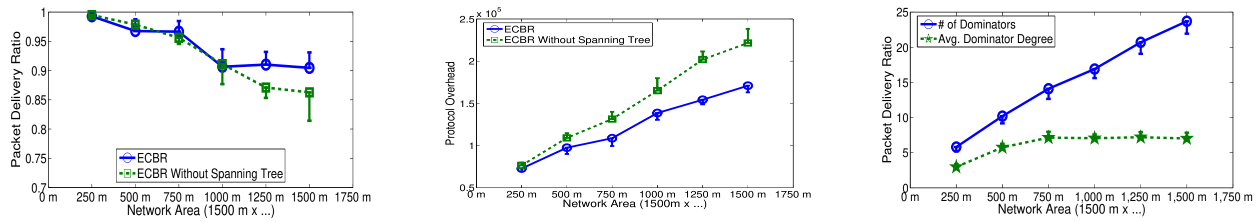
5.5.2 Network Area

Network area is another parameter that we held constant in the previous experiments. We vary it between 1500 m x 250 m and 1500 m x 1500 m in this section, and Figure 9 shows its impact; the other parameters are set to the same as in Section 5.1.1 except that the number of hot spots is kept at 60.

Figure 9 only shows ECBR's performance; the best of the other three protocols, BDSR, delivers 51% (std. dev. 13%) of the data packets in a 1500 m x 1500 m network. We can see that ECBR's performance gradually degrades from 99.3% to 90.5% when we increase the network area by six times.

This is because the backbone complexity grows as the network area increases. As shown in Figure 9(c), the number of dominator nodes and backbone links (# of Dominators X Avg. Dominator Degree) increases proportionally to the increase of network area. Therefore, the congestion impact of one route discovery packet worsens in a bigger network since it is broadcast on the backbone. Note that, even though the theoretical maximum dominator degree can be 47 [25], in practice it stays around 7 with little deviations as long as the backbone is not chain-like.

There are two ways to alleviate this problem. One is to use a spanning tree for route discovery on the backbone instead of broadcast, as ECBR does. A more sophisticated spanning-tree topology-management protocol might be our future work. Another is to make the



(a) Packet Delivery Ratio (b) Protocol Overhead (c) Backbone Complexity

Figure 9: Increasing Network Area

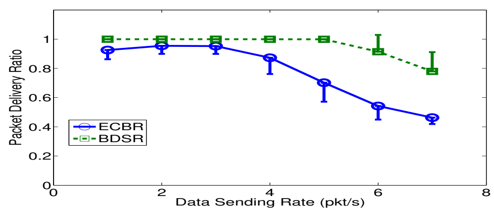


Figure 10: Increasing Data Sending Rate

backbone small, which has been the research focus of MCDS in recent years.

5.5.3 *Sending Rate (Bottleneck Issue)*

As a backbone approach, ECBR also suffers from the bottleneck problem. Figure 6 has shown us that ECBR's performance starts to drop after 50 concurrent CBRs. This section further investigates the impact of sending rate that we fixed to 1 pkt/s in the previous sections.

We vary the sending rate from 1 pkts/s to 7 pkts/s, and compare ECBR's performance to BDSR's in Figure 10. We use a static network and only 20 long-term CBR connections, each of which lasts 850 seconds. This setting greatly favors BDSR and makes the piggybacking and timestamping features of ECBR useless. We can see from Figure 10 that ECBR still shares the bottleneck problem with all the other backbone approaches. This suggests that backbone approaches are *not* suitable for long-term heavy-load network traffic. The source nodes should choose proper ad-hoc routing algorithms adaptively according to its traffic type.

6. 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 ECBR 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. Our algorithm significantly out performs the others when connections are short, the network is congested or mobility is high.

7. REFERENCES

[1] 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.

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

[3] 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.

[4] Z. J. Haas and M. R. Pearlman, "The performance of query control schemes for the zone routing protocol," in *SIGCOMM'98*, 1998, pp. 167–177.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] J. G. Jetcheva and D. B. Johnson, "Adaptive demand-driven multicast routing in multi-hop wireless ad hoc networks," in *MobiHoc '01: Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, Long Beach, CA, USA, 2001.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[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] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] C. Perkins, E. Royer, and I. Chakeres. (2004) Ad hoc on demand distance vector (aodv) routing. Internet-Draft, draft-perkins-manet-aodvbis-01.txt, Work in progress.

[34] C. K. Toh. (1999) Long-lived ad-hoc routing based on the concept of associativity. Internet-Draft, draft-ietf-manet-longlived-adhoc-routing-00.txt, Work in progress.

[35] 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.

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

[37] 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.