Load Balancing and Resource Reservation in Mobile Ad-Hoc Networks[1]

Gautam Chakrabarti Sandeep Kulkarni

Department of Computer Science and Engineering
Michigan State University

Abstract

To ensure uninterrupted communication in a mobile ad-hoc network, efficient route discovery is crucial when nodes move and/or fail. Hence, protocols such as Dynamic Source Routing (DSR) precompute alternate routes before a node moves and/or fails. In this paper, we modify the way these alternate routes are maintained and used in DSR, and show that these modifications permit more efficient route discovery when nodes move and/or fail. Our routing protocol also does load balancing among the number of alternate routes that are available. Our simulation results show that maintenance of these alternate routes (without affecting the route cache size at each router) increases the packet delivery ratio. We also show that our approach enables us to provide QoS guarantees by ensuring that appropriate bandwidth will be available for a flow even when nodes move. Towards this end, we show how reservations can be made on the alternate routes while maximizing the bandwidth usage in situations where nodes do not move. We also show how the load of the traffic generated due to node movement is shared among several alternate routes. In addition, we adaptively use Forward Error Correction techniques with our protocol and show how it can improve the packet delivery ratio.

Keywords: Ad-hoc Networks, Routing, Load balancing, Bandwidth Reservation, QoS.

[1]Email: {chakra10, sandeep}@cse.msu.edu

Web: `http://www.cse.msu.edu/~{chakra10,sandeep}`

Tel: +1-517-355-2387

A preliminary version of the paper appears in [1]. This work was partially sponsored by NSF CAREER CCR-0092724, DARPA Grant OSURS01-C-1901, ONR Grant N00014-01-1-0744, NSF equipment grant EIA-0130724, and a grant from Michigan State University.

1 Introduction

The recent emergence of mobile devices has increased the relevance of mobile ad-hoc networks. Such networks are formed by a collection of wireless nodes that are free to move about, often in a restricted amount of space. Such movement of nodes results in temporary networks, formed by a set of nodes due to their proximity to each other. Often, all mobile hosts in a network may not be in the transmission range of each other. In such scenarios, each node acts not only as a host sending (respectively, receiving) data to (respectively, from) another mobile host, but also as a router. Thus, nodes use multi-hop routes to reach their destinations. This task of routing data through multiple hops to the destination becomes all the more challenging due to the possibility of *router movement* in the middle of transmission.

To ensure uninterrupted communication in the presence of node movement, it is necessary to discover a new route efficiently. More specifically, if an intermediate node finds that it cannot reach the next hop on the route to the destination, then that intermediate node needs to find an alternate route to the destination. For uninterrupted communication, it is necessary that a new route be available as soon as a node becomes unreachable. In other words, it is necessary to identify alternate routes even before a node moves away or fails.

While identifying and using alternate routes, it is important to ensure that other flows using that alternate route are not affected and that appropriate bandwidth is available for the *rerouted* flow. However, one cannot simply reserve the entire requested bandwidth for a data transmission on the alternate routes, as it will lead to underutilization of the network bandwidth in the case where no nodes move. Based on the above discussion, in this paper, we focus on two conflicting goals that need to be met in ad-hoc networks: (1) ensuring the availability of an alternate route that provides the required bandwidth, and (2) maximizing the available bandwidth when no node moves/fails.

We proceed as follows: First we begin with the observation that in source-based routing when an intermediate node detects that it cannot reach the next hop in a source route, it can use precomputed alternate routes to other intermediate nodes mentioned in the packet header, to transmit that packet. This can be achieved by caching routes to other nodes in the network. If a node uses such a *route cache*, it is highly probable that an intermediate node may have an alternate route to other intermediate nodes in the source route even if it does not have an alternate route to the final destination. To this end, we aim to maximize the number of alternate routes cached at each node. At the same time, we need to be careful so as not to cache stale routes.

Second, we note that a node may have *multiple* alternate routes to reach another node in the network. Hence, a rerouted transmission needs to consume only a small amount of bandwidth from each individual alternate route, and thus can minimally affect other flows using links in the alternate route. This implies that

it is important to study the number of alternate routes to a particular node that are available in general, and to see how many of them can be used for rerouting. Availability of such valid alternate routes would help in using the available bandwidth effi ciently. It would also enable us to deal with the case where some alternate routes have become invalid due to node movement.

Third, we consider the effect of the rerouted flow on other flows in the network. As argued earlier, reserving the requested bandwidth in a single alternate route would result in ineffi cient bandwidth usage if there were no node movement/failure. Also, trying to reserve parts of the requested bandwidth along multiple alternate routes would generate high control overhead. Generating more control overhead may in effect be even more harmful since there is no guarantee how long the reserved alternate routes will remain valid. Moreover, attempting to reserve bandwidth when a packet is waiting to be rerouted would increase the end-to-end delay. Hence, we cannot afford to explicitly reserve bandwidth along the alternate route(s). At the same time, we need to have at least an *implicit* reservation along alternate routes to maximize the delivery ratio of rerouted packets. While having such an implicit reservation is benefi cial, it is also important to maximize the network bandwidth utilization. This implies that we need to have an implicit reservation such that (1) this reserved bandwidth is utilized to the maximum possible extent by rerouted packets, and (2) the rest of the network bandwidth is effi ciently utilized by ordinary data packets. Hence, we study in this paper how the amount of implicitly reserved bandwidth affects the packet delivery ratio.

Fourth, we consider the issue of reusing the existing route while determining alternate routes. The reuse of the existing route will enable us to use the bandwidth already reserved on that route. This will, in turn, remove the need of reservation teardown on the reused links. Also, if the number of new links on the alternate route is minimized then there is a greater potential that suffi cient bandwidth will be available on that alternate route. Moreover, as the existing route was *validated* recently, it is more likely to be valid (except for the next hop) than other cached routes.

Fifth, we analyze the effect of Forward Error Correction techniques on the behavior of our routing protocol. We consider the effects of sending parity packets for all blocks of data packets. This leads to the possibility of transmitting parity packets even when the data packets are transmitted successfully to the destination. This implies that too much bandwidth may be wasted by transmission of such parity packets. Hence, we analyze the issue of selectively sending parity packets *only* when the source receives an indication that some data packets might have been dropped. In such a scenario, the source can transmit parity packets for some recently transmitted data blocks.

In the context of the issues discussed above, this paper takes up the issues of *bandwidth reservation* and *route maintenance* in the face of node movement. Our approach provides better route availability between hosts and better delivery of data packets compared to the existing Dynamic Source Routing Protocol pre-

sented in [2]. We use the network simulator *ns* [3] to compare the performance of DSR and our protocol. We present simulation results showing the performance improvements we have achieved with our modifications.

Organization of the paper. The rest of the paper is organized as follows: In Section 2, we describe the DSR protocol for ad-hoc networks and the optimizations added to it. In Section 3, we present our approach, and in Section 4, we present simulation results comparing the performance of DSR and our protocol. Finally, in Section 5, we present related work and conclude in Section 6.

2 Dynamic Source Routing

In Dynamic Source Routing protocol for ad-hoc networks, the sender of a packet determines the route that the packet should follow in order to reach the destination. The entire *source route* is inserted in the packet header. A node receiving the packet determines from the header if it is an intermediate host in the route or if it is the final destination. If it is an intermediate node, it forwards the packet to the next hop as specified in the source route. If it is the final destination, the packet is instead delivered to the network layer. The protocol performs two major operations: *route discovery* and *route maintenance*.

Route discovery. When a host wants to send a packet, it consults its cache of previously discovered routes to determine if it has a route to the destination. If the host does not have a valid route, it broadcasts a *route request* packet containing its own address and the destination address for which it is requesting a route. Each host receiving the route request consults its cache to see if it has a route to the destination. If the cache does not yield a valid route, the node inserts its address in the packet header and broadcasts it again. Thus, the route taken by the packet gets stored in its header. If no intermediate host has a cached route and the destination is *reachable* from the source, the route request packet will ultimately reach the destination. When a packet reaches the destination or a node with a valid cached route, the node replies to the source with a *route reply* packet containing the source route discovered.

Route maintenance. DSR uses a hop-by-hop acknowledgment at the data link level to detect failure of a link. If a host is not able to reach the next hop while trying to transmit a packet, it sends a *route error* packet to the source of the packet giving addresses of nodes at both the ends of the failed link. The source host removes the hop from the cache and truncates routes containing that link at the failed point.

Previous Improvements to DSR. The Dynamic Source Routing protocol as presented in [4] has undergone some modifications in [2], some of which are:

- IEEE 802.11 requires an RTS/CTS/Data/ACK exchange for all unicast packets. This implies that data packets can be transmitted only through bidirectional routes. The source routing protocol is modified to use only bidirectional links for data transfer.

- Nodes are modified to work in *promiscuous* mode. A node in this mode overhears packets even if it is not listed in the source route. This approach allows nodes to learn about route failures by tapping *route error* packets. Moreover, if a node overhears a packet which has its own address listed in the unprocessed portion of the source route, it implies that the node is set to receive the packet through a longer route. In that case, it can let the source of that packet know about the available shorter route by sending a *gratuitous route reply*.

- When a node forwarding a packet to its next hop discovers that the node is unreachable either due to a link failure or a node movement/failure, it consults its cache to find an alternate route to the destination. If the node has another route to the destination, it changes the source route appropriately and forwards the packet according to this new route.

The version of DSR implemented by CMU Monarch [5] has a few additional modifications. Notably, when an intermediate node forwards a packet to its next hop, it snoops into the unprocessed portion of the route in the packet header to get a route to the destination. This route is cached by the intermediate node. The node can use this route for transmission of its own packets or for rerouting packets from another source if some existing route fails. This version of DSR has been ported to the Network Simulator, *ns* [3]. We compare our protocol with this implementation.

3 Proposed Improvements

In this section, we describe our approach for route maintenance and bandwidth allocation. First, we describe our approach to validate the hypothesis: when an intermediate node needs to find an alternate route due to node movement/failure, it should try to reuse the existing source route as much as possible. Then, in Section 3.1, we discuss our approach to load balancing. In Section 3.2, we address our modifications to the cache implementation. Subsequently, in Section 3.3, we discuss our approach for route reservation when a node starts communicating with a destination. We present our approach to bandwidth reservation on alternate routes in Section 3.4. Finally, we present our Forward Error Correction scheme in Section 3.5. The simulation results for our algorithms are presented in Section 4.

When an intermediate host in a source route cannot reach its next hop along the route, it looks up its cache for alternate routes to reach the destination. If it finds an alternate route, it modifies the source route accordingly and transmits the packet to the newly selected next hop. While selecting such alternate routes, our approach strives to maximize the part of the original route that is preserved in the new route. As mentioned in the Introduction, this helps in reducing interference with other flows and in increasing the probability of finding bandwidth on the alternate routes.

With this intuition, in our protocol, when a node detects that its next hop along a route is unreachable, it tries to find an alternate route to the node that lies at a distance of 2 hops (mentioned henceforth as *hop-2 neighbor*) along the route. An alternate route to the hop-2 neighbor would enable the intermediate node to remove the next hop neighbor from the route, but keep the rest of the route intact. It follows that, if successful, this leads to maximum reuse of the existing source route. If such an alternate route is not present in the cache, our protocol searches for routes to nodes farther away in the source route. We have implemented and tested three versions of our approach:

1. An intermediate node starts scanning the source route from its hop-2 neighbor towards the destination. For each node in this route, it searches its cache for an alternate route to that node. It uses the first alternate route that it obtains from its cache to appropriately modify the source route and to send the packet to the newly discovered next hop. Hence, the part of the route starting from the current node to the node to which it found an alternate route is changed. If there is no alternate route available to any of the intermediate nodes, but there is a route to the destination, then it results in an entirely new alternate route from the current node. In the worst case, if there is no alternate route to any of the nodes including the destination, the intermediate node drops the packet.

2. The intermediate node, in contrast to the previous approach, starts scanning the source route from the destination node towards its hop-2 neighbor. If the cache has an alternate route, the source route is modified as in the first approach.

3. The intermediate node first checks if it has an alternate route to the destination. If not, similar to the first approach, it searches hop-2 neighbor, hop-3 neighbor, and so on.

For all the versions, the routing protocol takes care *not* to include the next unreachable hop in the alternate routes used for rerouting. This is achieved by first removing all the routes from cache that contain the link from the current node to the unreachable next hop. We first conducted simulations to compare these approaches and to validate the hypothesis that reusing the existing routes is desirable. Observe that the first version follows this hypothesis. The second approach is a variant of the first, but it does not follow our intuition. The third approach attempts to find an alternate route only when DSR fails to find a route to a destination. Since we find that the first approach outperforms the others, we only present the first approach here.

3.1 Load Balancing

As we discussed in the Introduction, using alternate routes to reroute all packets of a flow may interfere with normal data transmissions through that alternate route. This problem is aggravated if multiple flows

facing route failures use the same alternate route to transmit their packets. Hence, our routing protocol does *load balancing* among the number of alternate routes that are available. The protocol uses multiple alternate routes (if available) in round robin order for rerouting packets that face a route failure. Our simulations have shown that in general nodes have two or more alternate routes to 90% of the nodes in the network. We use these routes for load balancing.

3.2 Modifications to the Cache

In our protocol, we modify the way cache is updated. The first modification deals with how the protocol learns of new routes, and the second modification deals with how the protocol uses its primary and secondary cache of routes. The total cache size is kept fixed to 64 entries for both our protocol and for DSR; this number was selected based on the experiments from [5].

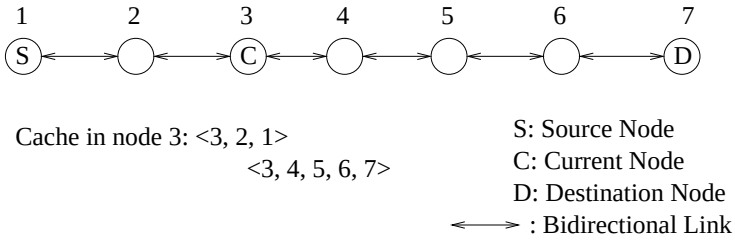


Figure 1: Learning of route segments from the source route in a data packet header

In DSR, a node caches a route when it receives a new route in reply to a *route request* packet, or when it *overhears* a packet not addressed to it and snoops into the source route to discover the route contained in that packet. Also, an intermediate node forwarding a packet snoops into the source route in the packet header and extracts the segment of the source route starting from the current node to the destination. Our protocol, in addition, extracts the route segment starting from the current node to the source of the packet. Thus, we can cache alternate routes for both the destination and the source (as shown in Figure 1) of the packet. Note that extracting the route segments in both directions from the current node in the source route is, however, possible only for data packets. This is because, the source route enclosed in the data packet header has been proved to be bidirectional in the route discovery phase. Such a route segment cannot be learnt from a route request packet since it has only traversed the links in one direction. For a route reply packet, the route segment from the current node towards the destination is bidirectional, and hence, can be cached. The route segment from the current node towards the source is still liable to be unidirectional, and hence, not extracted by the current node.

Like DSR, our protocol also maintains two separate fixed-sized caches of routes: a *primary* cache and a *secondary* cache. The primary cache is used to store routes returned in reply to *route request* packets. The secondary cache stores routes that have been learnt by other ways, for example, by snooping into the header of a packet while it is being forwarded. In general routes are added more frequently to the secondary cache, as a node is able to learn more routes from others' packets than the number of explicit *route replies* it receives. This implies that a route is more quickly eliminated from the secondary cache than it would if it were in the primary cache. Hence, when a source node uses a route from the secondary cache to transmit its *own* packets, DSR *promotes* the route from the secondary cache to the primary cache. To reduce the overhead of load balancing, our protocol, however, needs the routes to remain in the same order throughout their existence in cache. Moreover, as explained in the next section, only routes for which bandwidth has been allocated are stored in the primary cache. Hence, we do not promote routes from the secondary to the primary cache. This implies that our protocol uses only a small amount of primary cache, but needs a larger secondary cache. To reflect this requirement, we have reduced the size of the primary cache, and increased the size of the secondary cache while keeping the total cache size constant. In addition, while replacing a route from cache, we try to preserve routes that are currently in use. This approach also helps in removing stale routes. This is because, if a route is being used or has been used recently, it is highly probable that the route is still valid. On the other hand, if a route has not been used for long, it is quite probable that it has become stale in a relatively high mobility network. Thus, our protocol efficiently maintains the caches of reserved and alternate routes.

3.3 Route Reservation

Now we discuss our approach towards reserving bandwidth for original source routes and alternate routes. Our routing protocol aims to provide Quality of Service guarantees to source nodes initiating a data transfer. When a source wants to send data to a destination, it tries to reserve the requested amount of bandwidth along a source route before starting the data transmission. Towards this end, we assume that a host would be able to estimate the available bandwidth, using some link-level techniques. This may also need a close interaction between the link layer and the routing layer for the routing protocol to use this knowledge of bandwidth availability. While the issue of estimating total bandwidth is outside the scope of this paper, we refer the reader to [6, 7] for approaches on estimating total bandwidth.

Route discovery. In the QoS version of our protocol, when a source host initiates a *route discovery* phase to reach a particular destination, it is required to state the amount of bandwidth, it intends to consume, in its packet header. An intermediate node processing a route request packet checks to see if it has enough available bandwidth to be able to accept the request. If the node determines that it can support the requested flow, it re-broadcasts the route request packet. In addition, it inserts an entry in a *Flow Table* specifying the

source-destination pair and the reserved bandwidth. If multiple flows for the same source-destination pair need to be supported, an additional identifier can be stored in the *Flow Table* entry to uniquely represent a data flow. It is important to reserve the bandwidth on receiving a route request packet, and to verify it on seeing a corresponding route reply packet. If, instead, the initial reservation is postponed till receiving of a route reply packet, then an intermediate node may allow simultaneous route request packets to propagate towards their destinations irrespective of their reservation requests, increasing control overhead. This process of initial resource reservation continues until either the request reaches the destination, or some intermediate node that does not have enough available bandwidth. In the latter case, the intermediate node drops the packet.

As in DSR, once a route request reaches the destination, the node reverses the *route record* so far formed in the packet header, and retransmits the *route reply* packet back to the source. When an intermediate node receives the *route reply* packet, it flags the corresponding reservation entry in the *Flow Table* indicating this. In the event that the reservation entry is absent in the table (that is, it has been removed as explained later), the intermediate node drops the route reply packet.

As the QoS version of our protocol aims to reserve bandwidth along a route before starting a data transfer, it requires the route request packet to reach the destination. It does not allow any intermediate node to reply to the source with a cached route it may have for the destination. (DSR and non-QoS version of our protocol continue to be configured to use cached routes during route discovery.)

Reservation teardown. If any of the links in the route record were not bidirectional, the route reply would not reach the source. Hence, the source would only receive routes whose all links are bidirectional (required by IEEE 802.11 as discussed earlier in Section 2). Thus, although a node makes a reservation when a route request packet travels towards its destination, the route may not work out due to any of a variety of reasons such as link failure or due to the source deciding to use a different route. This implies that we need a mechanism for efficient teardown of reservations from nodes present in the route. Explicit removal of reservations would result in increased control packet overhead. Hence, we take the approach of implicit teardown of reservations. Nodes maintain timeout values to determine if a reservation should be kept any more. In our protocol, a reservation is *not* removed as soon as its timeout expires; it is removed if its timeout has expired and some new data flow is requesting a reservation for which the node does not have enough free bandwidth.

This approach of implicit teardown has another advantage over using control packets to explicitly remove reservations. If explicit mechanisms were used, a node in a failed route would be immediately signaled to remove the reservation. In our protocol, on the other hand, a reservation is removed after a timeout period. Although a source requesting bandwidth during this time interval may not get its share, this allows the

existing reservation to be used again by the same data flow. As explained earlier, while choosing alternate routes when a link fails, we aim to reuse the existing route (and hence, reuse the existing reservations) as much as possible. Hence, the new route may have some of the nodes in the original failed route. In such a scenario, the new route would start using the resource reservations existing on these nodes, and only unused reservations in any other nodes would be timed out.

We maintain three timeout values: *route reply timeout*, *data start timeout*, and *data timeout*. The *route reply timeout* is used to remove reservations for which the node received a route request packet, but has not seen a route reply packet. This may happen, for instance, if some node towards the destination could not provide the requested bandwidth, or if one of the links on that route is unidirectional. The *data start timeout* expires if a node has seen a route reply for a reservation, but has not yet received data from the source. This can happen if any of the links from the current node towards the source does not work out, or if the source chooses to use a different (probably shorter) route for the data transfer. The *data timeout* value is used to keep track of flows which have transmitted data, but this timeout value has elapsed since the last data packet was forwarded. This can occur, for example, due to node movement so that the currently used route fails, and the source needs to use a separate route.

Revisiting the cache modifications. In the QoS version of our protocol, we store routes returned in *route replies* in the primary cache. Thus, routes in the primary cache have bandwidth allocated for them. When a source node initiates a data transfer, our protocol allows it to look for routes *only* in the primary cache (and not in the secondary cache), since we want it to use a route for which bandwidth has been reserved. Hence, to support multiple data flows between the same source and destination, routes in the primary cache can also store the corresponding identifiers representing the data flows for which they are reserved. Thus, successive flows for the same source-destination pair would need to reserve bandwidth separately through a route request phase before they can transmit data. As in DSR, the shortest available route in the primary cache to the destination is used for data transmission. In the QoS version, routes in the secondary cache are used only for rerouting packets through alternate routes when the original route fails. (Note that DSR and the non-QoS version continue to use the routes in the secondary cache during route discovery.) Such an approach in our QoS protocol still allows the possibility of a source using a route whose reservation has been timed out from some intermediate node. In such a scenario, the intermediate node not having the required reservation would drop the data packets. Our simulation results show that this has only a minimal effect on packet delivery ratio.

3.4 Implicit Reservations on Alternate Routes

We make implicit reservations for rerouted flows. Consider the case where k disjoint alternate routes are available when any packet needs to be rerouted due to node movement/failure. In this case, $\frac{1}{k}^{th}$ of the flow will be transmitted on each alternate route. With this intuition, we allow a node to reserve only $\frac{k}{k+1}^{th}$ bandwidth while making reservations. The remaining $\frac{1}{k+1}^{th}$ ($\frac{1}{k}^{th}$ of $\frac{k}{k+1}^{th}$) bandwidth is implicitly reserved for rerouted flows. We found that in over 90% cases, two (or more) disjoint alternate routes are available when a flow needed to be rerouted. Hence, while simulating the QoS version of our algorithm, we let $k=2$. Thus, $\frac{1}{3}^{rd}$ bandwidth is implicitly reserved for rerouted flows. If the redundancy in an ad-hoc network is high, higher values of k can also be used; larger values of k will reduce the bandwidth that is reserved for rerouted flows.

The amount of implicitly reserved bandwidth needs to be chosen judiciously. This is because, if a relatively large fraction of the available bandwidth is reserved to be shared by alternate flows, then a part of this reservation may never be used by rerouted packets leading to reduced network utilization. On the other hand, too low a reservation may increase the loss of rerouted packets. We analyze these effects of varying amounts of implicit reservations on the packet delivery ratio in Section 4. It is to be noted that when a rerouted flow reuses parts of the original route, nodes present in the original route use the existing reservations as explained earlier, and the implicitly reserved bandwidth is used only by other nodes in the new route.

3.5 Forward Error Correction

Our results (as presented in Section 4) show that for many ad-hoc network environments, our QoS version gives a packet delivery ratio of 90% or more. In such scenarios, we feel that techniques such as forward error correction [8] can effectively provide a higher packet delivery ratio. In the simplest implementation of FEC, given two parameters m and n, $n > m$, for each group of m packets, $n-m$ additional parity packets are sent. If the receiver receives any m packets (some data packets and some parity packets), it can obtain all m data packets.

We have added an FEC(4,8) scheme to the QoS version of our protocol, to explore if we can improve the performance by sending a relatively low number of parity packets. In our implementation, we aim to minimize the bandwidth consumed by parity packets. To this end, for each group of m(=4) data packets, we do *not* send the corresponding parity packets immediately. Instead, when a source node receives a *route error* packet, it is highly probable that some of the most recent packets it had sent were dropped. Hence, on receiving a *route error* packet, the source node uses a separately available route to send parity packets for the

groups of data packets sent in the last $2 \times S$ seconds. S is a predetermined constant signifying the average packet delay between the point of failure and the source node. Thus, if the route fails at time t, and the route error packet reaches the source at time $(t + S)$, then the source should send parity packets for all data sent in the time interval $(t - S)$ to $(t + S)$. This is, of course, based on the assumption that the packet delay between the two points is approximately same in both directions. Once the parity packets for the blocks of data packets are sent, the source node continues to send data packets (without the corresponding parity packets). When the source node receives another route error packet, it again sends the corresponding parity packets. It is to be noted that when the source is transmitting parity packets, it may receive a route error packet signifying that some or all of the parity packets might have been dropped. In such a scenario, the source does not retransmit the parity packets. More specifically, our protocol does not attempt to send parity packets for the same block of data more than once. Our results show that when the packet drop rate is around 10% without this FEC implementation, a suitable number of parity packets for blocks of data dropped can reduce the loss rate to half.

We have also analyzed an FEC(4,8) scheme which transmits parity packets for *all* blocks of data packets. Interestingly, this scheme is not as effective as our selective FEC scheme. This is because such a scheme would transmit many unnecessary parity packets, that is, parity packets for data that have been transmitted to the destination successfully. In addition, this would consume a large fraction of the available bandwidth, thus affecting the delivery of data packets. Our selective FEC(4,8) scheme, on the other hand, saves bandwidth compared to the earlier version, and effectively transmits parity packets for data packets that might have been lost. We compare the performance of the two schemes in Section 4.

We also experimented our QoS protocol with an FEC(4,6) scheme. It was, however, less effective than our FEC(4,8) scheme. This is because, often a whole block of data packets is dropped due to a route failure. In such scenarios, the parity packets for an FEC(4,6) scheme would only waste bandwidth without being able to retrieve the data packets. Hence, in this paper, we present results for our FEC(4,8) schemes that transmit parity for 3 seconds and 9 seconds worth of data, respectively. It is to be noted that our FEC schemes also produced similar improvements when applied to the DSR protocol.

4 Performance Comparison

For our simulation, we use the network simulator *ns* (Version 2.1b8a) developed as part of the VINT project [3]. Protocols are evaluated with ad-hoc network topologies consisting of 50 wireless nodes, moving about in a rectangular space. The simulation time is 900 seconds for each run. The protocol takes as input a scenario file and a data traffic generation file. The scenario file specifies the movement of each node, and the traffic generation file has the data transfer characteristics giving details such as when each source node

starts a data transfer, the number of packets to be transmitted per second, and the size of each packet. The link bandwidth is 2 Mbit/sec for all the results.

We use the *random waypoint model* [4] to model node movement in our simulations. Each run of the protocol is characterized by a *pause time*. At the start of the simulation, each node remains stationary for *pause time* seconds. Then, each node selects a destination from the rectangular space randomly, and starts moving towards the target with a speed uniformly distributed between 0 and a maximum speed of 20 meters per second. At the destination, the node again stays there for *pause time* seconds before moving again. For small data rate (Figure 2), we use a rectangular space of dimensions 1500m $\times$ 300m. For large data rate (Figure 3 onwards), we use a space of dimensions 1800m $\times$ 1000m. In the QoS version of our protocol that we compare with DSR, 66% of the bandwidth is reserved for normal data flows and the rest is for rerouted flows; this value was chosen based on the observation that when messages needed to be rerouted, typically, two alternate routes were available.

We ran our simulations with networks containing CBR (constant bit rate) sources. First, we compute the effect of node movement on the percentage of packets dropped (= total number of packets dropped $\times$ 100 / total number of packets sent) (cf. Figure 2). The graph in Figure 2(a) is for a network of 10 data sources, while that in Figure 2(b) is for a network of 20 sources. Each source has a sending rate of 4 packets per second, with a packet size of 64 bytes. We conduct our experiments for the following values of pause time: 0, 30, 60, 120, 300, 600, and 900 seconds. A pause time of 0 means constant mobility, while that of 900 seconds implies no node movement. With high node mobility, the percentage of data packets dropped by the non-QoS version of our protocol is around half of that dropped by DSR. This performance improvement in the non-QoS version is obtained without any extra control packet overhead. This is due to the fact the the non-QoS protocol does not make reservations during route discovery; it only modifies the action taken when a route fails. Our QoS version introduces a small overhead due to the fact that a route request packet needs to reach the destination, and cannot be replied to by an intermediate node.

As shown in Figure 2(a), the QoS version is best when the number of sources is 10. This is due to the reason that routes used in the QoS version are more stable; during route reservation, the QoS version validates the route being used. However as number of sources is increased, due to the extra overhead of route reservation, our non-QoS version is better.

Table 1 shows the packet rate and the packet size for the data rates used in this paper. Results presented henceforth are for networks of 20 sources and 30 data flows. Figure 3 compares our QoS protocol with DSR for the percentage of data packets dropped as a function of the data rate of a source. The graph in Figure 3(a) is for a pause time of 600 seconds, while that in Figure 3(b) is for a pause time of 60 seconds. In our QoS protocol, a source sends data packets only after it gets a reservation. This only requires coordination

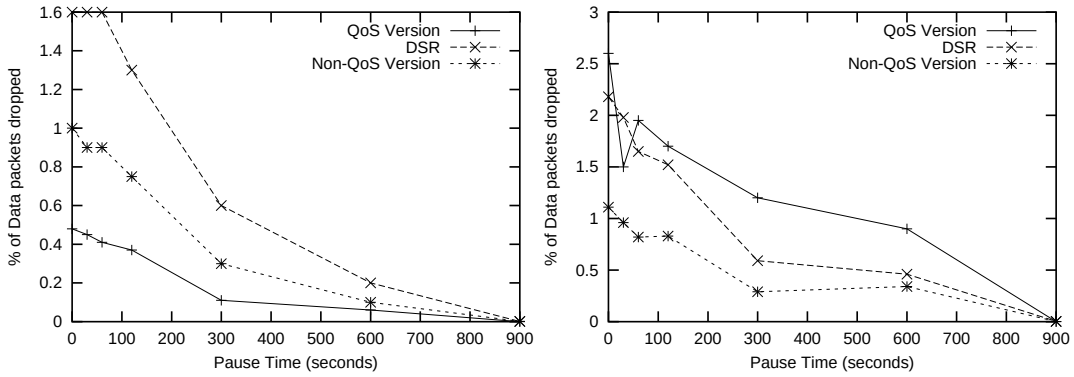
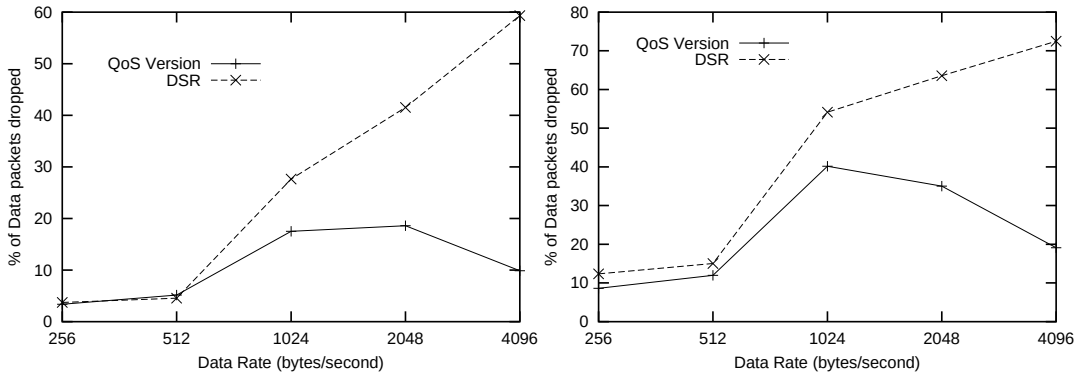


Figure 2: Comparison among the three protocols of the percentage of data packets dropped as a function of pause time. (a) Number of sources = 10, (b) Number of sources = 20

Data Rate (bytes/second)	Packet Rate (packets/second)	Packet Size (bytes)
256	4	64
512	4	128
1024	8	128
2048	8	256
4096	8	512

Table 1: Data Rate Specifications

between the application sending data and the routing protocol. This is because if an application needs QoS service, it has to get a resource reservation before it can start transmitting packets. For comparison, we modify DSR so that it does not send packets when the sender buffer is full. In other words, in simulating DSR, we ensure that packets are never dropped at the source due to route unavailability. As discussed earlier, the results in Figure 3 show that the performance improvement of our QoS protocol over DSR increases with the data rate. Even at high data rates of each source node transmitting at 4096 bytes/second, our protocol manages to deliver 80% to 90% of data packets transmitted, whereas DSR is able to deliver only around 30% to 40% of the data packets. It is also interesting to note in the graphs in Figure 3 that the performance of our QoS protocol actually improves while moving from a packet rate of 2048 bytes/second to that of 4096 bytes/second. This is because at such a high data rate, few flows actually get reservation and, hence, they can transmit most of their packets even through alternate routes in the event of a route failure. For lower data rates, many flows get reservations initially, but some of them are not able to reroute packets when a route fails.



(a) **(b)**

Figure 3: Comparison between the two protocols of the percentage of data packets dropped as a function of the data rate of a source node. (a) Pause time = 600 seconds, (b) Pause time = 60 seconds

While Figures 2 and 3 look at collective data loss, Figures 4 and 5 focus on the effect of data loss on individual flows. More specifically, Figures 4 and 5 plot the number of data flows that have the percentage of data packets dropped below a specific level. For example, the number of data flows that have their percentage drop between 0 and 10% (inclusive of 0) is plotted corresponding to 10 in the X-axis, the number of flows having percentage drop between 0 and 20% (inclusive of 0) is plotted corresponding to 20, and so on, until the number of flows having percentage drop between 90% and 100% (inclusive of both) is plotted corresponding to 100. Figures 4 and Figures 5 show the simulation results for 600 and 60 seconds pause time respectively, for different transmission data rates. For both pause times, the data rate is varied from 256 bytes/second to 4096 bytes/second.

These results also show that the performance improvement of our QoS-protocol is more for higher traffic rates. In Figures 4 (a) and 4 (b), both the QoS version of our protocol and the DSR protocol have similar performance. In Figure 4 (c), 16 flows have less than 20% drop ratio for our protocol, whereas there are only 11 flows in this range for the DSR protocol. For higher data rate of 4096 bytes/second, Figure 4 (d) shows that our protocol has 16 flows having percentage drop less than 10, while DSR does not have any flow in this range. Also for higher data rates and lower pause time (cf. Figure 5), our version shows marked improvement over DSR. For example, for a data rate of 4096 bytes/second in Figure 5 (d), there are 29 flows for our protocol that have a percentage drop less than 40, whereas DSR does not have any flow in this range.

As seen from the results, our protocol has large improvement over DSR for high data traffic rate. This is because for such scenarios, our protocol effectively uses its route reservations and, hence, can guarantee high packet delivery ratio for most of the reserved data flows. On the other hand, DSR tries to transmit packets in every flow, effectively achieving less packet delivery ratio for all the flows. Moreover, as shown

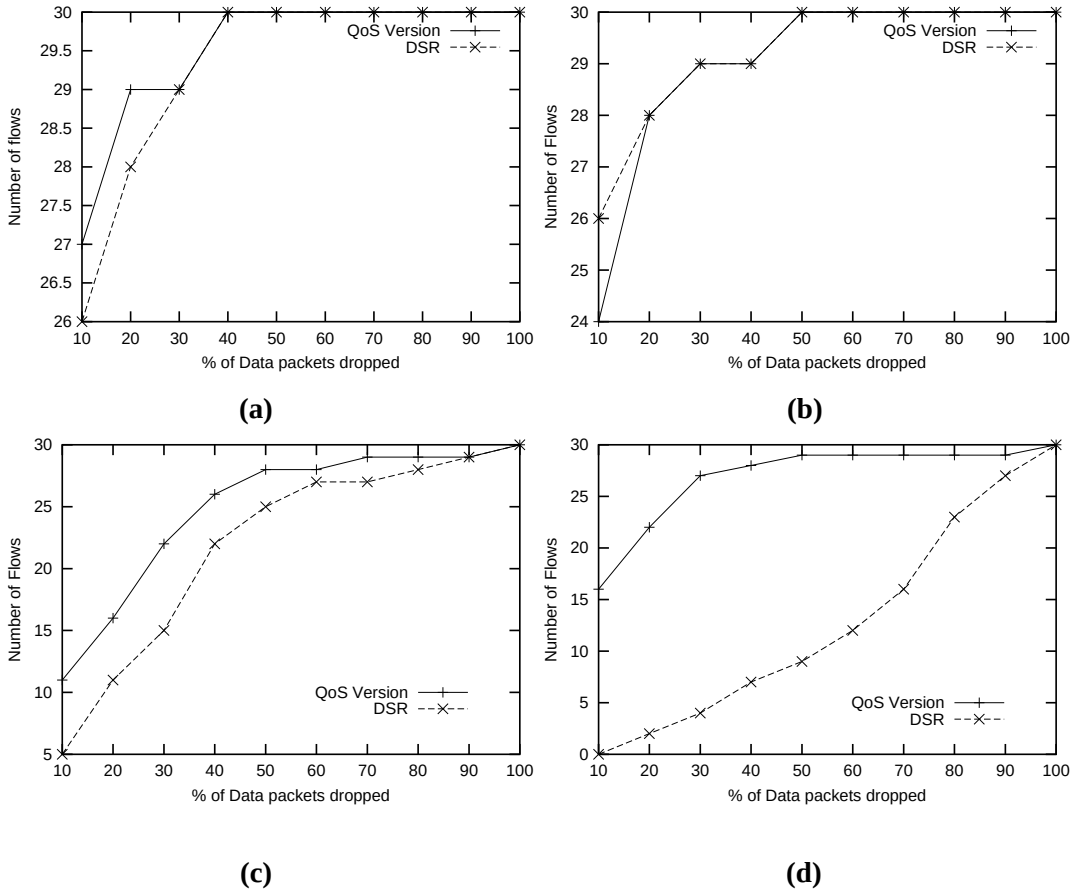


Figure 4: Cumulative distribution of the number of data flows having their percentage drop below a certain level, for a pause time of 600 seconds. (a) Data Rate = 256 bytes/sec, (b) Data Rate = 512 bytes/sec, (c) Data Rate = 1024 bytes/sec, (d) Data Rate = 4096 bytes/sec

in Figure 3, by allowing some flows (with reservations) to transmit data into the network (and disallowing other flows), we achieve a higher total packet delivery ratio compared to DSR.

Figure 6 compares different versions of our QoS protocol employing no FEC scheme, and FEC(4,8) schemes that send parity packets for all data packets, and for data packets sent in the last 3 seconds, and 9 seconds, respectively. As earlier, our QoS version in these results has an implicit reservation of 66% bandwidth for normal data flows, and the rest is for rerouted flows. The graphs, for 600 seconds and 60 seconds pause time respectively, show improved packet delivery ratio when FEC is employed. Moreover, the FEC scheme transmitting parity for 9 seconds worth of data performs better than that for 3 seconds. We have also experimented with schemes that send parity packets for data sent in the last 5 and 7 seconds respectively (results not presented). We have found that the packet delivery ratio increases gradually as

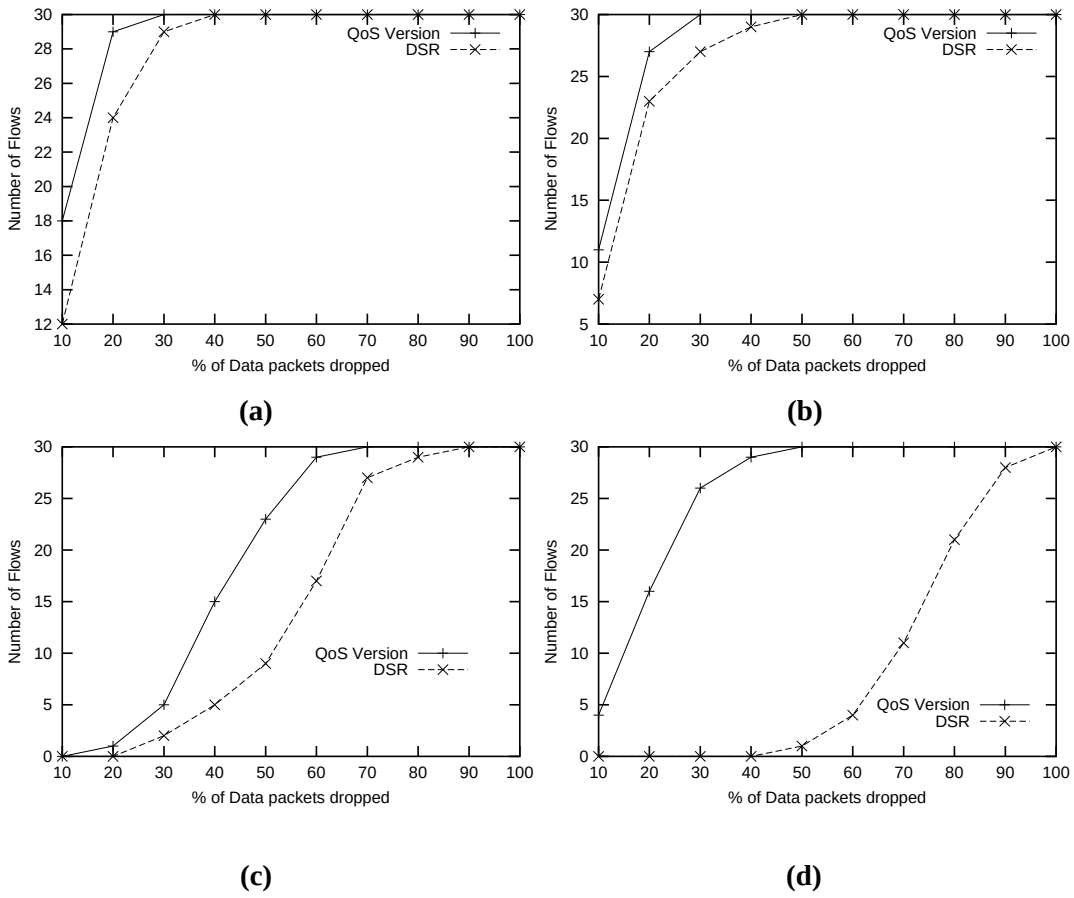


Figure 5: Cumulative distribution of the number of data flows having their percentage drop below a certain level, for a pause time of 60 seconds. (a) Data Rate = 256 bytes/sec, (b) Data Rate = 512 bytes/sec, (c) Data Rate = 1024 bytes/sec, (d) Data Rate = 4096 bytes/sec

we move from the FEC (3 seconds) scheme through the 5 and 7 seconds schemes to the FEC (9 seconds) scheme. This is because, as explained in Section 3.5, by selectively transmitting parity for data, our protocol effectively recovers much of the data lost due to route failures without consuming too much bandwidth. Moreover, any data packet lost during the time interval of $2 \times S$ seconds due to any other reason, also has the chance of being recovered by the transmitted parity packets. The results also show that even when parity is sent for all data packets, the packet delivery ratio is very close to that for FEC (3 seconds), and is less than that for FEC (9 seconds). Experiments conducted with a similar implementation of FEC on the original DSR protocol also gave similar amounts of improvement (cf. Figure 7) in terms of packet delivery ratio. From these experiments, we observe that if the movement is low (pause time = 600 in our experiments) FEC is especially useful with our QoS version as the loss rate typically reduces to acceptable limits (approximately 10% or less). By contrast, for the case where DSR is used or where movement is high, the loss rate remains high in spite of FEC. Thus, we suggest that FEC should be used in conjunction with our QoS protocol when node movement is (expected to be) less.

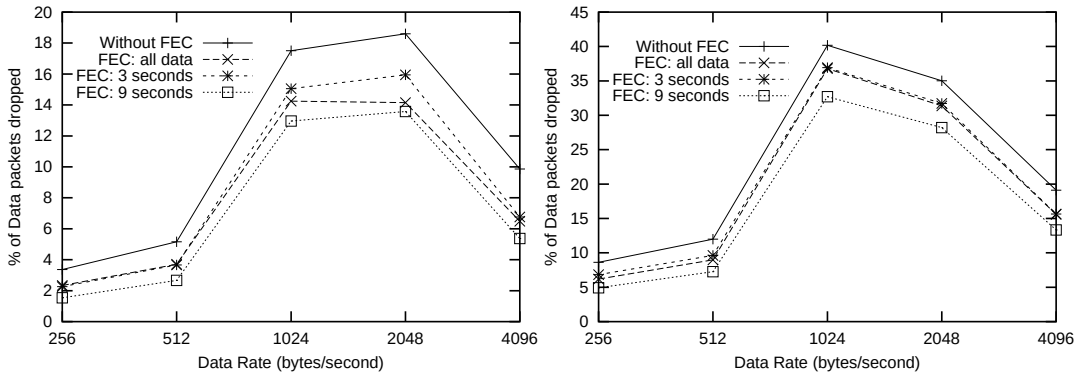


Figure 6: QoS Protocol: Percentage of data packets dropped for different number of groups of data packets for which FEC parity packets are sent. (a) Pause time = 600 seconds, (b) Pause time = 60 seconds

Figure 8 compares different versions of our QoS protocol having different amounts of implicit reservations for normal data flows. Each graph plots results for 50%, 60%, 66%, and 75% of bandwidth reservations for normal data packets, respectively. The packet delivery ratio is highest for 50% bandwidth reservation, and decreases while moving towards a reservation of 75%. When 75% of the bandwidth is reserved for normal data flows, many rerouted packets cannot get the required bandwidth, and are dropped. A 66% reservation provides the required bandwidth to rerouted packets in most scenarios, thus increasing the packet delivery ratio. Our analysis shows that this version of our protocol also has better network utilization than the other versions. For the versions with 50% and 60% implicit reservations respectively, the delivery ratio

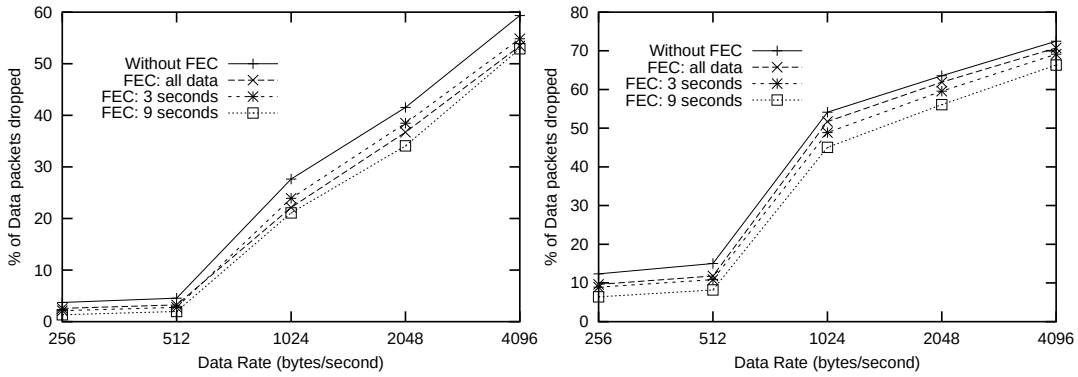


Figure 7: DSR Protocol: Percentage of data packets dropped for different number of groups of data packets for which FEC parity packets are sent. (a) Pause time = 600 seconds, (b) Pause time = 60 seconds

is increased at the cost of bandwidth wastage from the 50% and 40% reservation, respectively, for rerouted packets. More specifically, rerouted packets do not need the entire bandwidth reserved for them. Moreover, due to a lower reservation for normal data flows, source nodes are able to transmit lesser number of data packets into the network, thus reducing packet drops. Thus, we have found that our QoS version with 66% implicit reservation results in a relatively high packet delivery ratio and efficient network utilization. Hence, in this paper, we have compared this version with the DSR protocol.

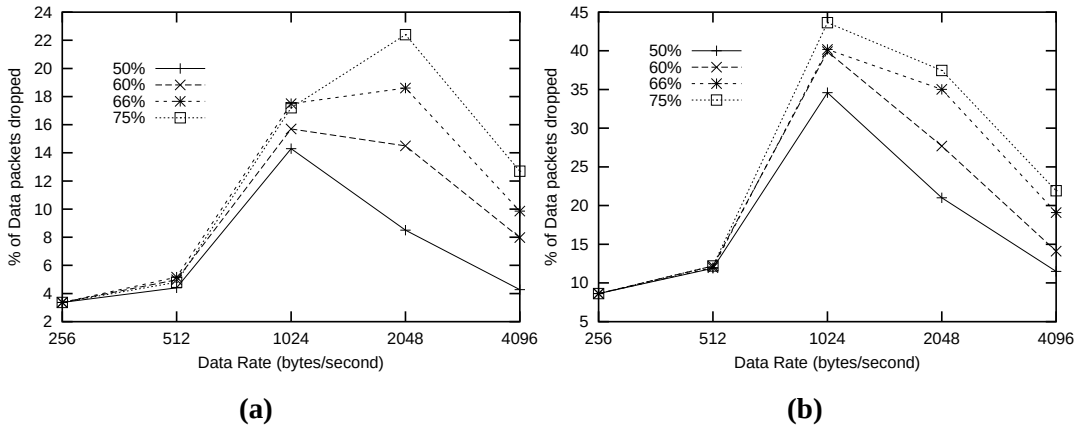


Figure 8: QoS Protocol: Percentage of data packets dropped for different amounts of bandwidth implicitly reserved for normal data flows. (a) Pause time = 600 seconds, (b) Pause time = 60 seconds

5 Related Work

Ad-Hoc routing protocols have been categorized [9] as Table-driven and Source-initiated on-demand. Table-driven routing protocols try to maintain a consistent view of the network using one or more routing tables. In the event of network topology changes, routing table updates are exchanged between nodes in the network. Some of the prominent protocols in this category are Destination-Sequenced Distance-vector (DSDV) Routing [10], Wireless Routing Protocol [11], and the QoS routing protocol presented in [12]. A problem often attributed to these protocols is that even in the absence of data traffic between hosts, nodes would continue to exchange broadcast messages to keep their routing tables updated.

Source-initiated on-demand routing protocols discover a route only when needed. The source node does a *route discovery*. Once a route has been obtained, it is inserted into packets flowing from the node dictating the route the packet should take. Some of the prominent protocols in this category are the Dynamic Source Routing Protocol [4], and the Neighborhood Aware Source Routing [13].

The Neighborhood Aware Source Routing (NSR) [13] protocol maintains information about the two-hop neighborhood of a node. A node, while forwarding a data packet, tries to take advantage of the information it has about the status of the two links along the source route in its 2-hop neighborhood. A node periodically broadcasts *hello* packets to its neighbors to notify its presence. These packets are also used to collect link-state information. In our protocol, however, we try to maintain information about the 2-hop neighborhood by using the information already contained in the control packets. Hence, we do not generate any extra control overhead. NSR uses the neighborhood information to detect a failed link, and reroute the packets after finding an alternate route. Our protocol, on the other hand, uses the neighborhood information to efficiently find alternate routes for rerouting packets (possibly to the 2-hop neighbor) when the next hop node moves and/or fails.

Alternate Path Routing (APR) [14] provides load balancing by distributing traffic among a set of diverse paths, and studies the impact of route coupling on APR's performance. In Dynamic Load-Aware Routing (DLAR) [15], nodes use their load information (the number of packets buffered in the interface) to select a route. This is in contrast to our protocol (and DSR) which uses the shortest route to transmit packets, and finds another available shortest route if the current route fails. In DLAR, the destination waits for an appropriate amount of time to learn all possible routes. Then, it sends a route reply choosing the least loaded route. Hence, the source may have to wait for a considerable amount of time before it is able to transmit data. Intermediate nodes also periodically attach their load information with data packets. On detecting congestion, the destination broadcasts a route request packet towards the source (reverse of normal route request). Load Sensitive Routing (LSR) [16] uses information about the local load as well as the load in the neighborhood to select a route. In LSR, the destination compares the current path load with the initial

load information, and starts a route request phase (as in DLAR) if it detects congestion. Load-Balanced Ad hoc Routing (LBAR) [17] uses the degree of nodal activity to select a route. The route discovery process consists of a forward phase and a backward phase similar to DLAR. In the forward phase, a "setup message" broadcasted by the source reaches the destination containing nodal activity information for nodes along the route. The destination starts the backward phase by selecting the best-cost path.

INSIGNIA [18] is a framework for QoS in ad-hoc networks. Unlike our QoS protocol, in [18], nodes state their maximum and minimum bandwidth requirements. Intermediate nodes maintain *soft-state* about reservations. Periodically the destination node provides feedback to the source giving status information about ongoing flows. A destination node may also ask a source node to stop transmitting packets. When a flow is rerouted to a node not having enough resources, the node provides best effort service to the flow. By contrast, in our QoS protocol, a fraction of available bandwidth is reserved in anticipation of rerouted flows.

CEDAR [19] is another QoS routing algorithm for ad-hoc networks. It uses a self-organizing *core* of nodes. A core node performs route computations on behalf of other nodes present in its domain. When a node wants to communicate to a destination, it contacts its dominator node in the core to find a route to the destination, which meets the QoS requirement. Nodes in the core use *core broadcast* to exchange network topology information with other nodes in the core. By contrast, our algorithm does not send any extra control packets.

SWAN [20] is a stateless network model for wireless ad-hoc networks. In contrast to our QoS approach, intermediate nodes in SWAN do not keep per-flow state information. The model provides service differentiation by using rate control for UDP and TCP best-effort traffic, and sender-based admission control for UDP real-time traffic. Similar to our route-discovery phase in which we establish an end-to-end reservation, a source node in SWAN has a *request/response* probe phase. During this phase, the source sends a *request* packet to estimate the end-to-end bandwidth availability between the source and the destination. Thus, this packet needs to reach the destination. In contrast to our approach, however, the packet does not carry the required bandwidth. The source, after getting the *response* packet from the destination decides if the required bandwidth is satisfied.

The approach for load balancing and route reservation is also applicable in other domains. Our approach has been used in an application that is a variation of the beam experiment used in [21]. In this problem, the network consisted of a simply supported elastic 2-D grid. Each node in the grid consisted of a sensor-actuator pair, where the sensor provided velocity measurements and the actuator applied force. This grid was subject to external vibrations and the sensors and actuators were used to minimize the vibration at all nodes. Due to the correlation between sensor values at multiple nodes, it was necessary to communicate sensor values of a node to other nodes in the network. By using our approach for load balancing and implicit bandwidth

reservations, it was found [22] that the quality of vibration control is maintained when nodes fail.

The transmission power of packets in ad-hoc networks is an important factor determining among other things the number of neighbors of a node, the bandwidth available to other nodes in the network, and the battery life [23] [24] [25]. SSA [26] discovers and maintains routes based on signal strength information available at the link level. A node keeps track of the signal strength of link layer beacons that it receives from its neighbors. Thus, it can detect *strongly connected* and *weakly connected* neighbors, and uses the strong links to route packets. The VMAC algorithm [27] is a Virtual MAC algorithm that runs in parallel to the MAC algorithm on a mobile host, and estimates MAC-level statistics related to service quality. Based on this estimation of service quality, a mobile host decides if a new flow with a specific service requirement should be admitted.

6 Conclusion and Future Work

In this paper, we focused on the problem of route maintenance and bandwidth allocation in ad-hoc networks. We presented two protocols, a QoS version and a non-QoS version. The main features of these protocols are as follows: (1) When the route specified by the source breaks due to a node movement/failure, that source route is reused –as much as possible– while rerouting the packets on an alternate route. (2) When the route specified by the source breaks, intermediate nodes use multiple alternate routes so that the rerouted flow does not interfere with other flows in the network. We find that in most cases, these alternate routes are disjoint and, hence, if k alternate routes are available then only $\frac{1}{k}^{th}$ bandwidth is used on each alternate route. (3) In the QoS version of our protocol, a source explicitly reserves the requested bandwidth before transmitting. For the case where the route chosen by the source breaks, no explicit reservations are made on the alternate routes used. However, implicit reservations are maintained on all links. More specifically, if k alternate routes are available for load balancing then $\frac{1}{k+1}^{th}$ bandwidth is used for rerouted flows and a node permits reservations for $\frac{k}{k+1}^{th}$ of the maximum limit. Thus, regular data packets can use a fraction of the total available bandwidth, while *only* rerouted packets use the rest. This implies that in absence of node movement, the bandwidth implicitly reserved for alternate routes would be left unused in the QoS version of our protocol. The simulation results in Section 4 show that by letting $k = 2$, such implicit reservation typically provides the required bandwidth. (4) A source node, on receiving a route error packet, transmits FEC parity packets only for blocks of data sent recently. Our simulation results show that the delivery ratio for our protocol is higher than that of DSR.

In our QoS protocol, a source node may initially obtain multiple reservations for the same data flow. Once the node settles with the shortest reserved route, the other reservations are timed out. Hence, this bandwidth would be unavailable to other flows that might try to do a route request during the timeout

interval. Our protocol tries to minimize this by choosing a relatively low timeout value.

The simulation results in Section 4 show that our protocols are better than DSR for both low and high data rates, as well as for networks with low or high node mobility. Our protocol sends FEC parity packets for selective blocks of data, thus minimizing the parity packet overhead and improving the delivery ratio. With low data rates, our protocol delivers close to 100% of transmitted data packets. With high data rates, our protocol effi ciently uses the available bandwidth and tries to maximize the number of flows having very low data loss rates. In addition, nodes in our protocol maintain a running average of data being transmitted for each source node. This implies that our protocol can verify if a source node is really transmitting at the requested rate. Hence, our protocol enables the intermediate nodes to charge the respective source nodes for the QoS service it provides.

There are several possible extensions to this work; in our simulations we used a predetermined amount of implicitly reserved bandwidth for alternate routes. The bandwidth requirement along alternate routes will, however, vary depending on factors like the number of alternate routes available, the transmission rate of the source nodes, and the number of competing data flows. A possible extension is to dynamically change the implicit reservation based on a combination of these factors. This may require increased control packet overhead in the form of information a node should share about its alternate routes and bandwidth requirements with other nodes in the network. In addition, our FEC scheme sends parity packets for data sent during a predetermined amount of time. This period of time can be adapted, probably based on estimated packet delays, to better reflect the actual number of data blocks for which parity needs to be sent.

References

[1] G. Chakrabarti and S. Kulkarni. A modified approach to dynamic source routing in mobile ad-hoc networks. *Proceedings of the 1st International Conference on AD-HOC NetwOrks and Wireless (ADHOC-NOW), Toronto, Canada*, September 2002.

[2] J. Broch, D. A. Maltz, D. B. Johnson, Y. C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. *Proceedings of the 4^{th} Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'98), Dallas, Texas, USA*, October 1998.

[3] Kevin Fall and Kannan Varadhan. editors, The ns manual. November 1997. The VINT Project, UC Berkeley, LBL, USC/ISI, Xerox PARC, Available at: http://www.isi.edu/nsnam/ns/.

[4] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. *Mobile Computing*, 5:153–181, 1996.

[5] The CMU monarch project: Wireless and mobility extensions to ns-2. http://monarch.cs.cmu.edu/cmu-ns.html.

[6] Samarth H. Shah, Kai Chen, and Klara Nahrstedt. Dynamic bandwidth management for single-hop ad hoc wireless networks. *First IEEE International Conference on Pervasive Computing and Communications (PerCom'03), Fort Worth, Texas*, March 2003.

[7] Saverio Mascolo, Claudio Casetti, Mario Gerla, M. Y. Sanadidi, and Ren Wang. Tcp westwood: Bandwidth estimation for enhanced transport over wireless links. *Proceedings of the seventh annual international conference on Mobile computing and networking, Rome, Italy*, pages 287–297, 2001.

[8] R. E. Blahut. *Theory and Practice of Error Control Codes*. Addison-Wesley, Reading, MA, 1983.

[9] E. M. Royer and C-K Toh. A review of current routing protocols for ad-hoc mobile wireless networks. *IEEE Personal Communications Magazine*, pages 46–55, April 1999.

[10] C. E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. *Computer Communications Review*, pages 234–244, October 1994.

[11] S. Murthy and J. J. Garcia-Luna-Aceves. An efficient routing procotol for wireless networks. *ACM Mobile Networks and Applications Journal, Special Issue on Routing in Mobile Communication Networks*, pages 183–197, October 1996.

[12] Chunhung Richard Lin and Jain-Shing Liu. QoS routing in ad hoc wireless networks. *IEEE Journal on Selected areas in Communications*, 17(8), August 1999.

[13] Marcelo Spohn and J.J. Garcia-Luna-Aceves. Neighborhood aware source routing. *Proceedings of ACM MobiHoc, Long Beach, California*, October 2001.

[14] M. R. Pearlman, Z. J. Haas, P. Sholander, and S. S. Tabrizi. On the impact of alternate path routing for load balancing in mobile ad hoc networks. *Proceeding of 2000 First Annual Workshop on Mobile and Ad Hoc Networking and Computing, Mobihoc 2000, Boston, MA, USA*, pages 3–10, August 2000.

[15] S. J. Lee and M. Gerla. Dynamic load-aware routing in ad hoc networks. *Proceeding of IEEE International Conference on Communications*, pages 3206–3210, June 2001.

[16] Kui Wu and Janelle Harms. Load-sensitive routing for mobile ad hoc networks. *Proceedings of 10^{th} IEEE International Conference on Computer Communications and Networks (ICCCN '01), Phoenix, Arizona*, pages 540–546, October 2001.

[17] Hossam Hassanein and Audrey Zhou. Routing with load balancing in wireless ad hoc networks. *Proceedings of the 4th ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems, Rome, Italy*, pages 89–96, 2001.

[18] Seoung-Bum Lee and Andrew T. Cambell. INSIGNIA: In-band signaling support for QoS in mobile ad hoc networks. *Proceedings of 5^{th} International Workshop on Mobile Multimedia Communications MoMuc '98, Berlin*, October 1998.

[19] Raghupathy Sivakumar, Prasun Sinha, and Vaduvur Bharghavan. CEDAR: a core-extraction distributed ad hoc routing algorithm. *IEEE Journal on Selected areas in Communication*, 17(8), August 1999.

[20] Gahng-Seop Ahn, Andrew T. Campbell, Andras Veres, and Li-Hsiang Sun. SWAN: Service differentiation in stateless wireless ad hoc networks. *Proceedings of IEEE INFOCOM'2002, New York, New York*, June 2002.

[21] A. Ledeczi, M. Maroti, and I. Bartok. Simple NEST application simulator. Technical report, Institute for Software Integrated Systems, 2001. Also available at http://www.isis.vanderbilt.edu/projects/nest/index.html.

[22] A. Arora, M. Gouda, T. Herman, S. Kulkarni, and M. Nesternko. Self-stabilization in networked embedded software technology (NEST). Available at: `http://www.cse.msu.edu/~sandeep/presentations/darpa_nest_feb_02/`, February 2002.

[23] Elizabeth M. Royer, P. Michael Melliar-Smith, and Louise E. Moser. An analysis of the optimum node density for ad hoc mobile networks. *Proceedings of the IEEE International Conference on Communications, Helsinki, Finland*, June 2001.

[24] L. Kleinrock and J. Silvester. Optimum transmission radii for packet radio networks or why six is a magic number. *Proceedings of the IEEE National Telecommunications Conference*, pages 4.3.1–4.3.5, December 1978.

[25] Tamer A. ElBatt, Srikanth V. Krishnamurthy, Dennis Connors, and Son Dao. Power management for throughput enhancement in wireless ad-hoc networks. *2000 IEEE International Conference on Communications (ICC 2000), Los Alamitos, CA*, 3:18–22, June 2000.

[26] Rohit Dube, Cynthia D. Rais, Kuang-Yeh Wang, and Satish K. Tripathi. Signal stability based adaptive routing (SSA) for ad-hoc mobile networks. *IEEE Personal Communications*, pages 36–45, February 1997.

[27] A. Veres, A. T. Campbell, M. Barry, and L-H. Sun. Supporting service differentiation in wireless packet networks using distributed control. *IEEE Journal of Selected Areas in Communications (JSAC), Special Issue on Mobility and Resource Manegement in Next-Generation Wireless Systems*, 19(10):2094–2104, October 2001.