Wireless Networks 8, 275–288, 2002


# Analysis of TCP Performance over Mobile Ad Hoc Networks

GAVIN HOLLAND *
*Department of Computer Science, Texas A&M University, College Station, TX 77843-3112, USA*

NITIN VAIDYA
*Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA*

**Abstract.** Mobile ad hoc networks have attracted attention lately as a means of providing continuous network connectivity to mobile computing devices regardless of physical location. Recent research has focused primarily on the routing protocols needed in such an environment. In this paper, we investigate the effects that link breakage due to mobility has on TCP performance. Through simulation, we show that TCP throughput drops significantly when nodes move, due to TCP's inability to recognize the difference between link failure and congestion. We also analyze specific examples, such as a situation where throughput is zero for a particular connection. We introduce a new metric, *expected throughput*, for the comparison of throughput in multi-hop networks, and then use this metric to show how the use of explicit link failure notification (ELFN) techniques can significantly improve TCP performance.

**Keywords:** mobile ad hoc networks, wireless networks, transport protocols, performance evaluation, explicit feedback, TCP

## 1. Introduction

With the proliferation of mobile computing devices, the demand for continuous network connectivity regardless of physical location has spurred interest in the use of mobile ad hoc networks. A mobile ad hoc network is a network in which a group of mobile computing devices communicate among themselves using wireless radios, without the aid of a fixed networking infrastructure. Their use is being proposed as an extension to the Internet, but they can be used anywhere that a fixed infrastructure does not exist, or is not desirable. A lot of research of mobile ad hoc networks has focused on the development of routing protocols (e.g., [10,11,13,16,18,20,22, 25,27,29–35]). Our research is focused on the performance of TCP over mobile ad hoc networks.

Since TCP/IP is the standard network protocol stack on the Internet, its use over mobile ad hoc networks is a certainty. Not only does it leverage a large number of applications, but its use also allows seamless integration with the Internet, where available. However, earlier research on cellular wireless systems showed that TCP suffers poor performance in wireless networks because of packet losses and corruption caused by wireless induced errors. Thus, a lot of research has since focused on mechanisms to improve TCP performance in cellular wireless systems (e.g., [2,3]). Further studies have addressed other network problems that negatively affect TCP performance, such as bandwidth asymmetry and large round-trip times, which are prevalent in satellite networks (e.g., [4,12]).

In this paper, we address another network characteristic that impacts TCP performance, which is common in mobile ad hoc networks: link failures due to mobility. We first present a performance analysis of standard TCP over mobile ad hoc networks, and then present an analysis of the use of explicit notification techniques to counter the affects of link failures.

* Corresponding author.

## 2. Simulation environment and methodology

The results in this paper are based on simulations using the *ns* network simulator from Lawrence Berkeley National Laboratory (LBNL) [14], with extensions from the MONARCH project at Carnegie Mellon [5]. The extensions include a set of mobile ad hoc network routing protocols and an implementation of BSD's ARP protocol, as well as an 802.11 MAC layer and a radio propagation model. Also included are mechanisms to model node mobility using precomputed mobility patterns that are fed to the simulation at run-time. For more information about the extensions, we refer the reader to [5]. Unless otherwise noted, no modifications were made to the simulator described in [5], beyond minor bug fixes that were necessary to complete the study.

All results are based on a network configuration consisting of TCP-Reno over IP on an 802.11 wireless network, with routing provided by the Dynamic Source Routing (DSR) protocol and BSD's ARP protocol (used to resolve IP addresses to MAC addresses). These are wildly used and studied protocols, and are likely candidates for implementation in commercial ad hoc networks. Hence, understanding how they work together, and with TCP, can lead to modifications that improve performance. Since we frequently refer to details of the DSR protocol, in the next paragraph we give a brief primer on DSR to familiarize the reader with its characteristics and terminology.

The Dynamic Source Routing (DSR) protocol was developed by researchers at CMU for use in mobile ad hoc networks [6]. In DSR, each packet injected into the network contains a routing header that specifies the complete sequence of nodes on which the packet should be forwarded. This route is obtained through *route discovery*. When a node has a packet to send for which it does not have a route, it initiates route discovery by broadcasting a *route request*. This request is propagated through the network until it reaches a node, say $x$, that knows of a route to the destination. Node $x$ then sends a *route reply* to the requester with the new route formed from the route at node $x$ concatenated with the source route in the request. To limit how far a request is propagated, a time-to-live (TTL) field is attached to every request along with a unique request identifier. A node that receives a *route request* that it has seen before, or that has lived beyond its time-to-live, drops the request. To reduce the number of route discoveries, each node maintains a cache of routes that it has learned. A node may learn of a route through route discovery, or through other means such as snooping routes in route replies and data packets, or eavesdropping on local broadcasts. This cache is updated through *route error* messages. *Route error* messages are sent by a node when it discovers that a packet's source route is invalid. The route discovery protocol, as implemented in the CMU extensions to *ns*, has two phases: a local broadcast (a *ring-0* search) followed by a propagating search. The *ring-0* search is initiated in the hope that a route can quickly be found in a neighbor's cache. If a route is not found within a small amount of time, a propagating search is attempted. If this fails, the protocol backs off and tries again, eventually giving up if a route is not found. This procedure repeats until all of the packets queued for that particular destination are dropped from the queue, or a route is found. A packet may be dropped from the queue if a route has not been found within a prespecified amount of time (the "Send Buffer Timeout" interval), or if the queue is full and newly arriving packets force it out. Route discoveries for the same destination are limited by the backoff and retry procedure, which is initiated per destination (versus per packet). Thus, regardless of the number of packets that need a route to the same destination, only one route discovery procedure is initiated. Once a route is found and a packet is sent, there is the possibility that the route becomes "stale" while the packet is in flight, because of node mobility (a route is "stale" if some links on the route are broken). In such an instance, DSR uses a mechanism called *packet salvaging* to reroute the packet. When a node $x$ detects that the next link in a packet's route is broken, it first sends a route error message to the node that generated the packet's route to prevent it from sending more packets on the broken route. Node $x$ then attempts to *salvage* the packet by checking its cache to see if it knows of another route to the packet's destination. If so, node $x$ inserts the new source route into the packet and forwards it on that route; if not, the packet is dropped.

We chose to keep most of the parameters of the simulations identical to those in [5], with a few exceptions. The following is a discussion of our simulation setup.

Our network model consists of 30 nodes in a $1500 \times 300\ m^2$ flat, rectangular area. The nodes move according to the *random waypoint* mobility model. In the random waypoint model, each node $x$ picks a random destination and speed in the rectangular area and then travels to the destination in a straight line. Once node $x$ arrives at its destination, it pauses, picks another destination, and continues onward. We used a pause time of 0 so that each node is in constant motion throughout the simulation. All nodes communicate with identical, half-duplex wireless radios that are modeled after the commercially available 802.11-based WaveLan wireless radios, which have a bandwidth of 2 Mbps and a nominal transmission radius of 250 m. TCP packet size was 1460 bytes, and the maximum window was eight packets.

Unless otherwise noted, all of our simulation results are based on the average throughput of 50 scenarios, or *patterns*. Each pattern, generated randomly, designates the initial placement and heading of each of the nodes over the simulated time. We use the same pattern for different mean speeds. Thus, for a given pattern at different speeds, the same sequence of movements (and link failures) occur. The speed of each node is uniformly distributed in an interval of $0.9v$–$1.1v$ for some mean speed $v$. For example, consider one of the patterns, let us call it $I$. A node $x$ in $I$ that takes time $t$ to move from point $A$ to point $B$ in the 10 m/s run of $I$ will take time $t/2$ to traverse the same distance in the 20 m/s run of $I$. So, $x$ will always execute the exact same sequence of moves in $I$, just at a proportionally different rate. See [19] for more details on the mobility patterns.

## 3. Performance metric

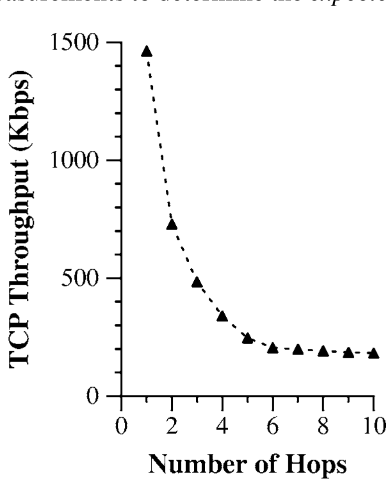
In this performance study, we set up a single TCP-Reno connection between a chosen pair of *sender* and *receiver* nodes and measured the throughput over the lifetime of the connection. We use throughput as the performance metric in this paper.

The TCP throughput is usually less than "optimal" due to the TCP sender's inability to accurately determine the cause of a packet loss. The TCP sender assumes that all packet losses are caused by congestion. Thus, when a link on a TCP route breaks, the TCP sender reacts as if congestion was the cause, reducing its congestion window and, in the instance of a timeout, backing-off its retransmission timeout (RTO). Therefore, route changes due to host mobility can have a detrimental impact on TCP performance.

To gauge the impact of route changes on TCP performance, we derived an upper bound on TCP throughput, called the *expected throughput*. The TCP throughput measure obtained by simulation is then compared with the expected throughput.

We obtained the *expected throughput* as follows. We first simulated a *static* (fixed) network of $n$ nodes that formed a linear chain containing $n-1$ wireless hops (similar to the "string" topology in [17]). The nodes used the 802.11 MAC protocol for medium access. Then, a one-way TCP data transfer was performed between the two nodes at the ends of the linear chain, and the TCP throughput was measured between these nodes. This set of TCP throughput measurements is analogous to that performed by Gerla et al. [17], using similar (but not identical) MAC protocols.

Figure 1 presents the measured TCP throughput as a function of the number of hops, averaged over ten runs. Observe that the throughput decreases rapidly when the number of hops is increased from 1, and then stabilizes once the number of hops becomes large. The primary reason for this trend is due to the characteristics of 802.11. Consider the simple four hop network shown in figure 2. In 802.11, when link 1–2 is active only link 4–5 may also be active. Link 2–3 cannot be active because node 2 cannot transmit and receive simultaneously, and link 3–4 may not be active because communication by node 3 may interfere with node 2. Thus, throughput on an $i$ hop 802.11 network with link capacity $C$ is bounded by $C/i$ for $1 \leq i \leq 3$, and $C/3$ otherwise. The decline in figure 1 for $i \geq 4$ is due to contention caused by the backward flow of TCP ACKs. For further explanation of this trend, we refer the reader to [17]. Our objective here is only to use these measurements to determine the *expected throughput*.




| Hops | Throughput (Kbps) |
|---|---|
| 1 | 1463.0 |
| 2 | 729.0 |
| 3 | 484.4 |
| 4 | 339.9 |
| 5 | 246.4 |
| 6 | 205.2 |
| 7 | 198.1 |
| 8 | 191.8 |
| 9 | 185.3 |
| 10 | 182.4 |

Figure 1. TCP-Reno throughput over an 802.11 fixed, linear, multi-hop network of varying length (in hops).

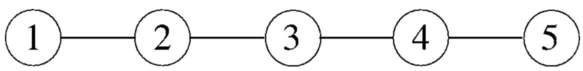



Figure 2. A simple multi-hop network.

The *expected throughput* is a function of the mobility pattern. For instance, if two nodes are always adjacent and move together (similar to two passengers in a car), the expected throughput for the TCP connection between them would be identical to that for 1 hop in figure 1. On the other hand, if the two nodes are always in different partitions of the network, the expected throughput is 0. In general, to calculate the expected throughput, let $t_i$ be the duration for which the shortest path from the sender to receiver contains $i$ hops ($1 \leqslant i \leqslant \infty$). Let $T_i$ denote the throughput obtained over a linear chain using $i$ hops. When the two nodes are partitioned, we consider that the number of hops $i$ is $\infty$ and $T_\infty = 0$. The expected throughput is then calculated as

$$\textit{expected throughput} = \frac{\sum_{i=1}^{\infty} t_i \cdot T_i}{\sum_{i=1}^{\infty} t_i}. \qquad (1)$$

Of course, $\sum_{i=1}^{\infty} t_i$ is equal to the duration for which the TCP connection is in existence. The measured throughput may never become equal to the expected throughput, for a number of reasons. For instance, the underlying routing protocol may not use the shortest path between the sender and receiver. Also, equation (1) does not take into account the performance overhead of determining new routes after a route failure. Despite these limitations, the expected throughput serves as a reasonable upper bound with which the measured performance may be compared. Such a comparison provides an estimate of the performance degradation caused by host mobility in ad hoc networks.

## 4. Measurement of TCP-Reno throughput

Figure 3(a) reports the measured TCP-Reno throughput and the expected throughput as a function of the mean speed of movement.

Note that the expected throughput is independent of the speed of movement. In equation (1), when the speed is increased, the values of $t_i$ for all $i$ becomes smaller, but the

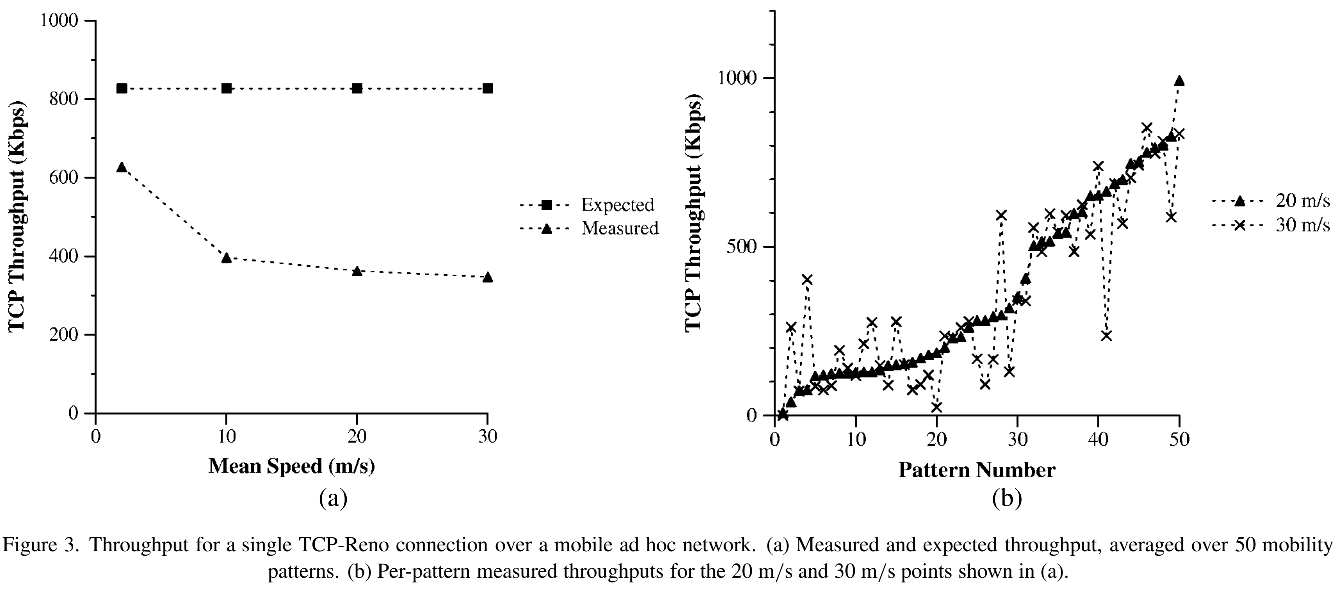



Figure 3. Throughput for a single TCP-Reno connection over a mobile ad hoc network. (a) Measured and expected throughput, averaged over 50 mobility patterns. (b) Per-pattern measured throughputs for the 20 m/s and 30 m/s points shown in (a).

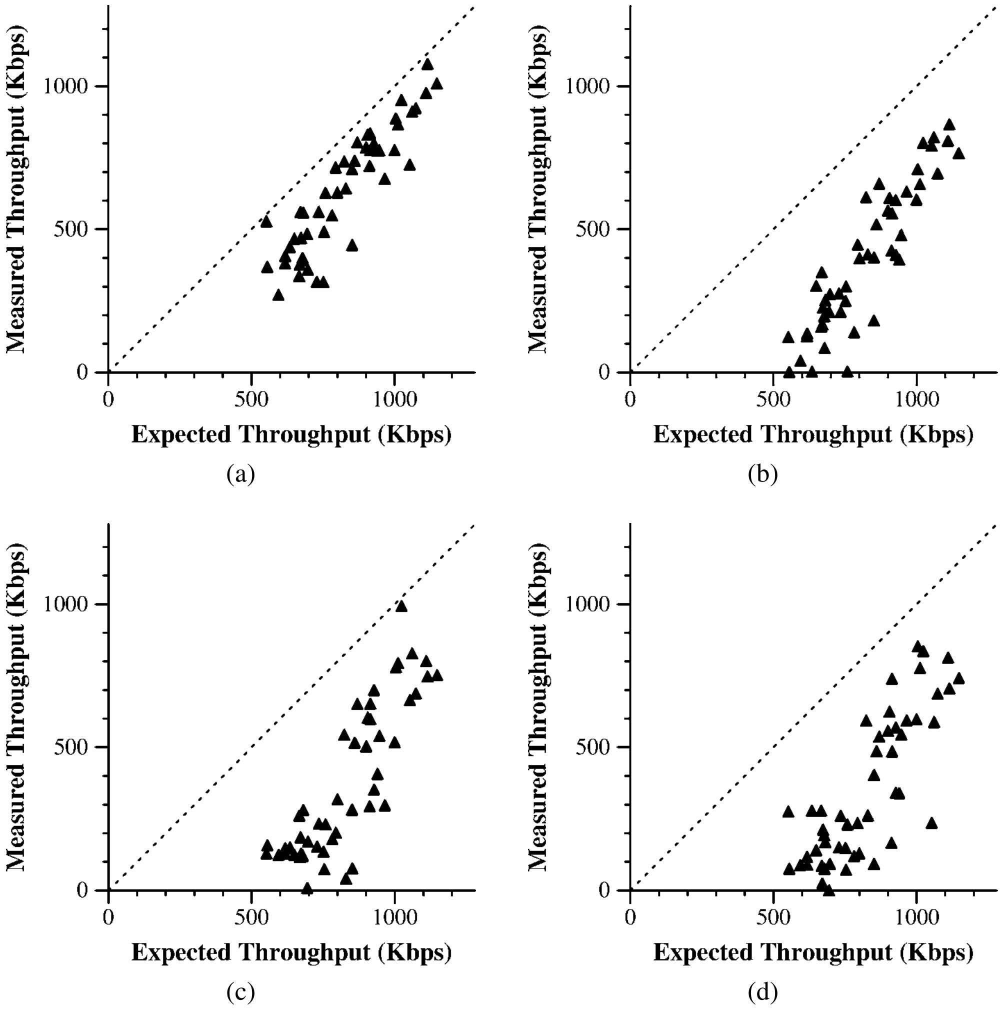



Figure 4. Comparison of measured and expected throughput for the 50 mobility patterns. Speed (in m/s) is (a) 2, (b) 10, (c) 20, and (d) 30.

ratio $t_i/t_j$ for any $i$ and $j$ remains the same. Therefore, the expected throughput for a given mobility pattern, calculated using equation (1), is independent of the speed.

Intuition suggests that when the speed is increased then route failures happen more quickly, resulting in packet losses, and frequent route discoveries. Thus, intuitively, TCP throughput should monotonically degrade as the speed is increased. In figure 3(a), the throughput drops sharply as the mean speed is increased from 2 m/s to 10 m/s. However, when the mean speed is increased from 10 m/s to 20 m/s and 30 m/s, the throughput averaged over the 50 runs decreases only slightly. This is a counter-intuitive result. The reason can be attributed, in part, to the network layer's problems maintaining routes for paths longer than a few hops at the higher node speeds. Thus, beyond a certain speed, throughput was commonly achieved only in the situation where the sender and receiver were within a few hops of each other. Another contributing factor to this result is that, under certain circumstances, throughput could potentially *increase* with speed. Consider, for example, figure 3(b), which plots the throughput for each of the 50 mobility patterns for the 20 m/s and 30 m/s mean speeds used in our simulations (the patterns are sorted, in this figure, in the order of their throughputs at 20 m/s). Observe that, for certain mobility patterns, the throughput increases when the speed is increased. This can happen, for instance, when fortuitous timing of TCP and MAC retransmissions, with regard to the state of the network (e.g., the position of the nodes in the network), results in the re-establishment of the packet flow at the higher speed but not at the slower speed. Section 5 discusses this anomaly in more detail.

Figure 4 provides a different view of the TCP throughput measurements. In this figure, we plot the measured throughput versus expected throughput for each of the 50 mobility patterns. The four graphs correspond to each of the four different mean speeds of movement. Because the expected throughput is an upper bound, all the points plotted in these graphs are below the diagonal line (of slope 1). When the measured throughput is closer to the expected throughput, the corresponding point in the graph is closer to the diagonal line, and vice versa. The following observations can be made from figure 4:

- Although, for any given speed, the points may be located near or far from the diagonal line, when the speed is increased the points tend to move away from the diagonal, signifying a degradation in throughput. Later in this paper, we show that, using a TCP optimization, the cluster of points in this figure can be brought closer to the diagonal.

- On the other hand, for a given speed, certain mobility patterns achieve throughput close to 0, although other mobility patterns (with the same mean speed) are able to achieve a higher throughput.
- Even at high speeds, some mobility patterns result in high throughput that is close to the expected throughput (for instance, see the points close to the diagonal line in figure 4(c) and (d)). This occurs for mobility patterns in which, despite moving fast, the rate of link failures is low (as discussed earlier, if two nodes move together, the link between them will not break, regardless of their speed).

Section 5 provides explanations for some observations made based on the data presented in figures 3 and 4.

## 5. Mobility induced behaviors

In this section, we look at examples of mobility induced behaviors that result in unexpected performance. The measured throughput of the TCP connection is a function of the interaction between the 802.11 MAC protocol, the ARP protocol, the DSR routing protocol, and TCP's congestion control mechanisms. As such, there are likely to be several plausible explanations for any given observation. Here, for each observation, we give one such explanation that we have been able to confirm using the measured data.

### *5.1. Some mobility patterns yield very low throughput*

We present one observed scenario wherein loss of some TCP data and acknowledgment packets (due to route failures) results in zero throughput. Note that we measure throughput as a function of the amount of data that has been *acknowledged* to the sender. In the example scenario discussed here, no acknowledgments are received by the sender during the 120 s lifetime of the TCP connection (the average speed for this case is 30 m/s). However, the *expected throughput* for the mobility pattern in this run is 694 Kbps. A path exists between the TCP sender and receiver nearly the entire time.

A condensed version of the simulation packet trace is shown in table 1. This trace was obtained with node 1 as the TCP sender and node 2 as the TCP receiver. In the table, the *Evnt* column lists the event type – s denotes that a packet is sent, r denotes that a packet is received, and D denotes that a packet is dropped. The *Resn* column lists the reason why a packet is dropped – NRTE means that the routing protocol could not find a route and END means the simulation finished. The Node, *SeqNo*, and *Pkt* columns report the node at which the event occurred, the TCP sequence number[1] of the packet depicted in the event, and the type of packet, respectively.

Table 1
Packet trace for a 30 m/s run that experienced zero throughput.

| *Evnt* | Time (s) | Node | *SeqNo* | *Pkt* | *Resn* |
|---|---|---|---|---|---|
| s | 0.000 | 1 | 1 | tcp | |
| D | 0.191 | 5 | 1 | tcp | NRTE |
| s | 6.000 | 1 | 1 | tcp | |
| r | 6.045 | 2 | 1 | tcp | |
| s | 6.145 | 2 | 1 | ack | |
| D | 6.216 | 21 | 1 | ack | NRTE |
| s | 18.000 | 1 | 1 | tcp | |
| s | 42.000 | 1 | 1 | tcp | |
| s | 90.000 | 1 | 1 | tcp | |
| D | 120.000 | 15 | 1 | tcp | END |
| D | 120.000 | 16 | 1 | tcp | END |
| D | 120.000 | 25 | 1 | tcp | END |

In this scenario, the sender and the receiver node are initially six hops apart and stay within six hops of each other for all but 6 s of the 120 s simulation. For 6 s, the network is partitioned, with the sender and receiver nodes being in different partitions.

Soon after the first packet is sent by node 1, a link break occurs along the route that causes a partition in the network. The partition causes the first packet to be dropped (at time 0.191 s) by the routing protocol on node 5, which was the forwarding node that detected the link failure. Eventually, the TCP sender on node 1 times out and retransmits the packet (at time 6.000). On the second attempt, the packet reaches the receiver, node 2, who sends a delayed acknowledgment (at time 6.145). However, the acknowledgment is sent on a route from node 2's cache that is stale (i.e., some links on the route are broken), so the acknowledgment is later dropped (at time 6.216). The remaining attempts to retransmit the packet also fail because of stale cached routes. In each instance, the packet is held by the ARP layer of a forwarding node until the end of the simulation (see the rows with *Evnt* = D and *Resn* = END in table 1). Each ARP layer is left holding a packet because its attempts to resolve the IP address of the next node in the route to a MAC address fail because of mobility.

Therefore, the TCP sender is unable to receive any acknowledgment from the receiver.

### *5.2. Anomaly: Throughput increases when speed is increased*

In the example discussed in this section, TCP throughput improves by a factor of 1.5 when the speed is increased from 10 m/s to 20 m/s. In the scenario under consideration, the TCP sender and receiver were able to reach each other 100% of the time, and spent 74% of the time at most two hops away. The nodes were never more than three hops away.

The characteristics of the connection between the TCP sender and receiver can be seen in the mobility pattern profile

[1] These are sequence numbers assigned by *ns* to TCP packets. *ns* does not number each octet individually; instead, the packets are numbered sequentially as 1, 2, etc. All references to TCP sequence numbers in this paper are the *ns* assigned sequence numbers.

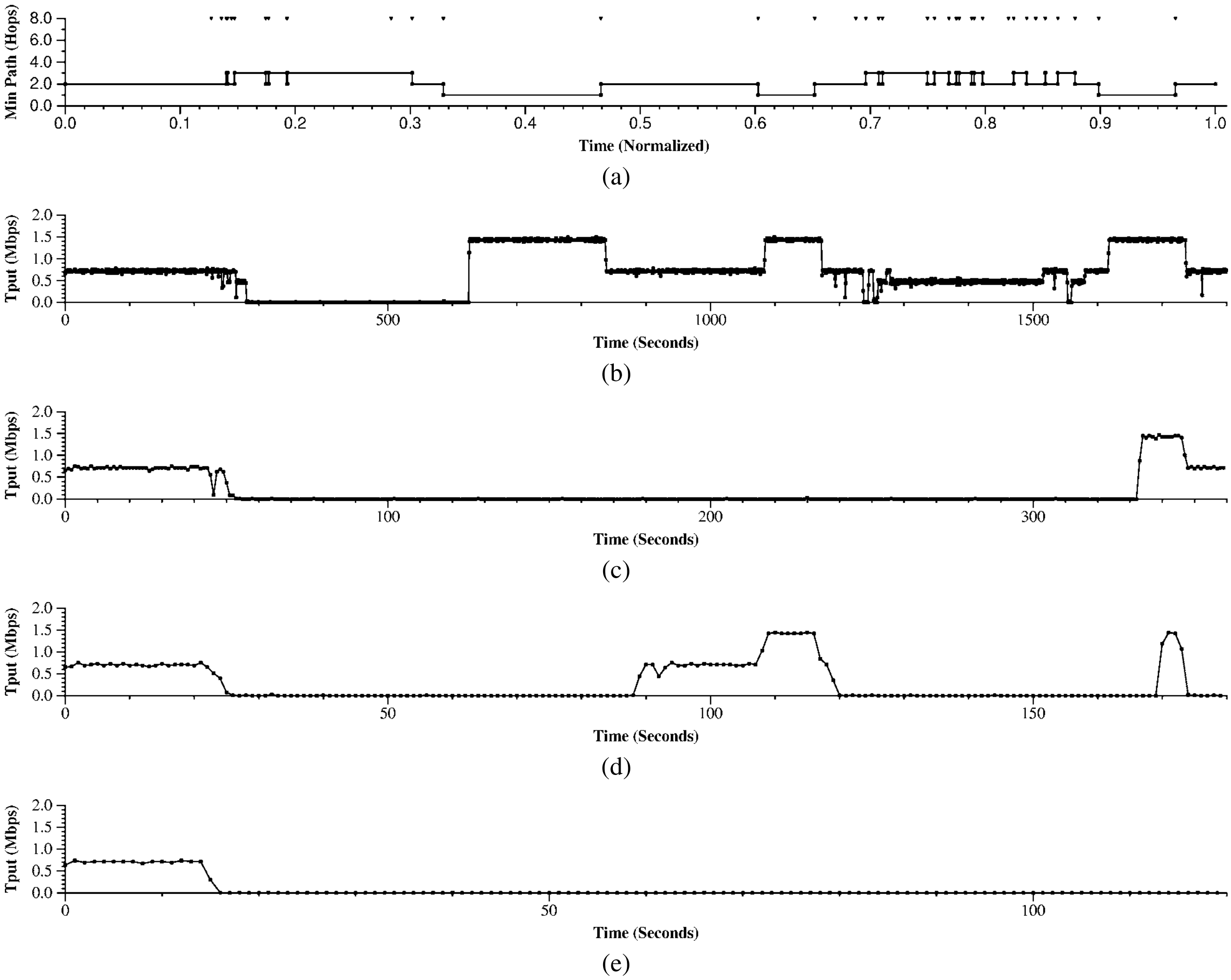



Figure 5. TCP-Reno performance for mobility pattern No. 20, demonstrating that an increase in mean node speed may result in an increase in mean throughput. (a) Mobility pattern profile. Mean speed (in m/s) is (b) 2, (c) 10, (d) 20, and (e) 30. The ticks at the top of (a) denote changes on the minimum path between the TCP sender and receiver. The curves in (b)–(e) show the measured throughput for the connection, averaged over 1 s intervals.

shown in figure 5(a) (see [19] for similar details on all of the patterns). The ticks shown at the top of the profile mark the points in the pattern at which the minimum path between the TCP sender and receiver changed. The curve shows the minimum path length (distance) in hops between the sender and receiver for the duration of the pattern. Notice that a change in the minimum path is not always caused by a change in path length (e.g., at the 0.28 mark in figure 5(a)), because the nodes on the path may change even though the total number of hops stays the same.

The other curves in figure 5 show the mean throughput over the TCP connection (averaged over 1 s) for each of the four mean node speeds. Note that, as mentioned in section 2, the sequence of moves that each node makes is identical, regardless of the mean speed.

The only difference is that a distance covered by a node, say $x$, over time $t$, such as in figure 5(b), takes $x$ a time of $t/2$ to cover in figure 5(c). This is analogous to a movie in which the time taken to show the same number of frames at rate $r$ takes half the time to show at rate $2r$. Thus, the mobility pattern profile shown in figure 5(a) can be used as a reference point for the other curves in figure 5. Note that the variations in the throughput for curves (b)–(e) are correlated to the path length in (a) because of the effect shown in figure 1, which we discussed earlier. Also note that DSR does not always use the minimum path when one is available, as seen around the 1450 s mark of figure 5(b).

*Discussion of figure 5(c).* In the 10 m/s run, the routing protocol uses symmetric forward and reverse routes (of optimal length) between the TCP sender and receiver for the first 50 s of the simulation, resulting in good initial throughput.

However, the sequence of path changes around the 50 s mark causes the TCP sender to back off, from which it fails to recover, until the final 30 s of the simulation. The details of the packet activity around the moment at which the initial backoff occurs is shown in figure 6. Leading up to the failure, the forward and reverse routes are symmetric and optimal in length (two hops). Around the 50.4 s mark, the route breaks (because of mobility) at the link between the intermediate node and the TCP receiver. This results in the queuing of nearly a full window of packets at the intermediate node. The intermediate node salvages the queued packets, then suc-

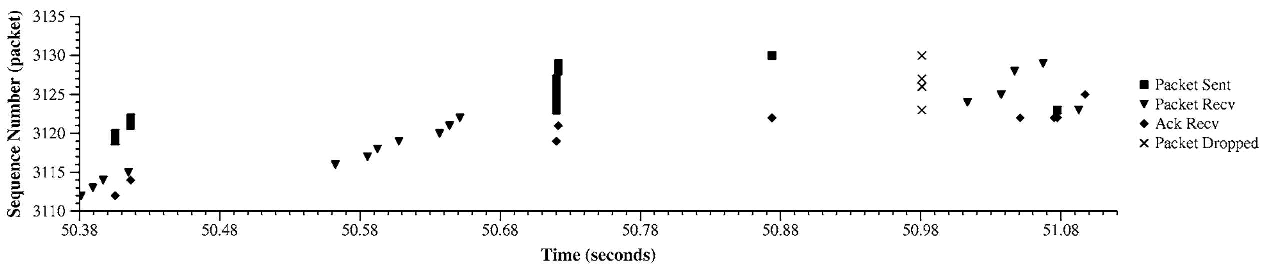



Figure 6. Detailed packet plot showing the beginning moments, around the 50 s mark in figure 5(c), at which a sequence of path changes, shown in figure 5(a), causes TCP to repeatedly time out and back off. Packet Sent and Packet Recv indicate the time at which a TCP data packet with the indicated *ns* sequence number was sent by the sender and arrived at the receiver, respectively, Ack Recv indicates the time at which a TCP acknowledgment was received by the sender with the indicated sequence number, and Packet Dropped indicates the time at which a data packet with the indicated sequence number was dropped.

cessfully delivers them to the receiver on a new forward route (seen around the 50.58 s mark). After detecting the failed link, the receiver chooses a new reverse route for sending acknowledgments, which is different than the forward route. However, the reverse route that it chooses is also stale, so several acknowledgments are lost before salvaging results in the arrival of two of the acknowledgments at the TCP sender around the 50.72 s mark. These acknowledgments trigger a burst of packets from the sender, which are immediately queued by the forwarding node at the next hop in the path, because, although the reverse route is good, the forward route is now broken by mobility. Another acknowledgment arrives later (around the 50.87 s mark), resulting in the queuing of another packet. Meanwhile, the forwarding node, which now has the full window queued, repeatedly tries to salvage the packets. This finally results in the loss of half of the packets (around the 50.98 s mark) by ARP, which fails to determine the MAC address of the node over the next hop in the salvaged route because the node has moved away. However, half of the packets are successfully salvaged on an alternate route and delivered (seen between the 51.0 s and 51.08 s marks), generating a sequence of duplicate acknowledgments (*dupacks*) from the receiver signifying the packet loss. After the third and fourth dupacks arrive, the TCP sender enters fast recovery and retransmits the lost packet (at the 51.08 s mark), but the lost packets cause the sender to timeout. The retransmission of the lost packet by the sender results in a brief burst of packets (seen as a small bump at the 51 s mark), but the routes break quickly thereafter, as the path changes from two to three hops, resulting in lost packets that cause the sender to timeout again.

For all subsequent timeouts, except one, stale routes result in packet losses even though the TCP sender and receiver are never more than three hops distance from each other. The one exception occurs around the 333 s mark, at which time a retransmitted packet results in the reestablishment of packet flow when the nodes are one hop away.

*Discussion of figure 5(d).* The 20 m/s run shares many of the characteristics of the slower 10 m/s run, but results in higher throughput because a retransmission late in the pattern (around the 90 s mark) succeeds in briefly reestablishing the flow of packets. Initially, the data flow is quickly stalled (around the 25 s mark) because of the loss of a full window of packets, which is caused by the same sequence of link changes in the pattern that affected the 10 m/s run. The throughput, again, degrades when repeated route failures induce packet losses, causing the TCP sender to time out and backoff. However, unlike the 10 m/s run, the packet flow is reestablished later in the pattern (at the 88 s mark) when a retransmitted packet results in the discovery of a good route when the nodes are only two hops apart. This success is why the 20 m/s run is able to transfer data at 1.5 times the rate of the 10 m/s run, for the same mobility pattern.

### *5.3. Summary and observations*

In this section, we present a summary of the effects of mobility on TCP performance that we observed in the previous examples and in our other experiments.

From the previous examples, it is clear that the characteristics of the routing protocol have a very significant impact on TCP performance. Most notable were the problems caused by the caching and propagation of stale routes. Even in relatively slowly changing topologies, the inability of the TCP sender's routing protocol to quickly recognize and purge stale routes from its cache resulted in repeated routing failures. Allowing intermediate nodes to reply to route requests with routes from their caches complicated this problem, because they often responded with stale routes. This was further amplified by the fact that other nodes could overhear or snoop the stale routes in the replies as they were propagated, spreading the bad information to caches in other nodes. We saw the effects of this problem in our simulations. For instance, in the simulation run presented in our first example (section 5.1), the TCP sender tried to use the same stale route three times because it received the route repeatedly from other nodes. In the latter two tries, the stale route came to the TCP sender by way of salvaging. The stale route that was used was a two hop route between the TCP sender and receiver. In each of the two instances, a neighboring node salvaged a packet from the TCP sender using the stale route, which the node had stored in its route cache. The neighboring node then sent the packet on the next hop in the salvaged route, back to the TCP sender. The result was that the TCP sender ended up trying to forward its own packet on a route that it had earlier determined was stale. However, we believe that these problems can potentially be solved using more effective cache maintenance strategies, including simple techniques like dynamically ad-

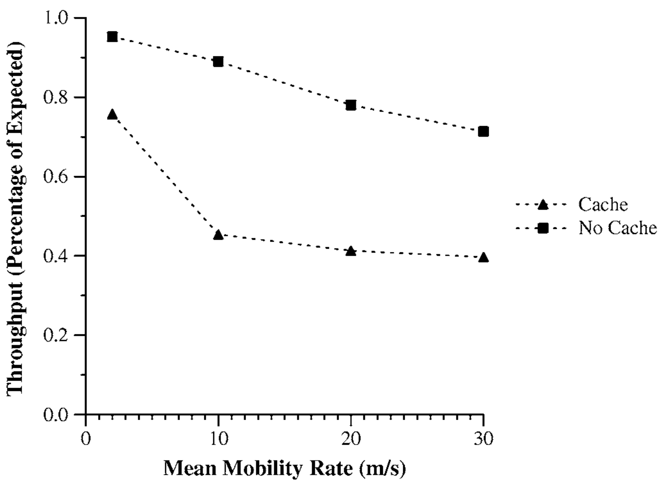



Figure 7. A comparison of TCP-Reno performance when DSR route replies from caches are, and are not, allowed.

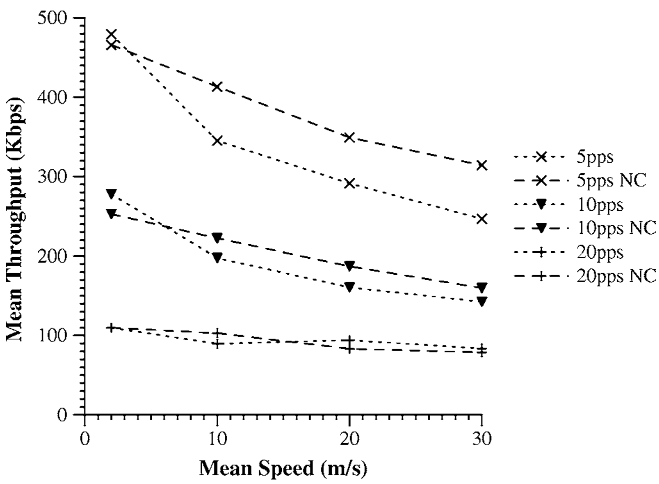



Figure 8. A comparison of TCP-Reno performance when DSR route replies from caches are, and are not (NC), allowed, and additional traffic is in the network: 10 CBR connections, each sending 5, 10, and 20 packets per second.

justing the route cache timeout mechanism depending on the observed route failure rate, the use of negative route information (mentioned in [6]), or the use of signal strength information.

Alternatively, replying from caches can be turned off altogether. This has a startling improvement in performance, as shown in figure 7. However, these results are for a single TCP connection in a network with no other data traffic. In a network with multiple data sources, the additional routing traffic introduced when replies from caches are not used could degrade performance.

Therefore, we simulated the same TCP connection in a network containing multiple CBR data sources to gauge the impact of the additional routing overhead when replies from caches are not allowed. These results are shown in figure 8. We looked at three different levels of network traffic using ten CBR connections across eight nodes (not including the TCP sender or receiver), each sending 512 byte packets at mean rates of 5, 10, and 20 packets per second (pps). Start times were staggered. For low mobility in the presence of CBR traffic, disallowing route replies from caches results in slightly lower TCP performance than when route replies from caches are allowed. This is due, in part, to the impact of the additional routing overhead. For moderate-to-high mobility, however, we see a consistent improvement in performance, which is clearly evident in the 5 pps and 10 pps curves. We observed that this improvement occurred because the steady traffic provided by the CBR connections increased the accuracy of cached routes by steadily exercising routes in the network, facilitating quick detection of broken links and, summarily, the purging of stale routes. However, for increasing levels of traffic the performance improvement decreases due to the additional routing traffic, until, for the 20 pps curve, disallowing route replies results in worse performance at 5 and 10 m/s, and only slight improvement at 20 and 30 m/s.

We also looked at a scenario where multiple TCP connections share the network. This data is presented in figure 9, which shows the mean throughput over five TCP connections

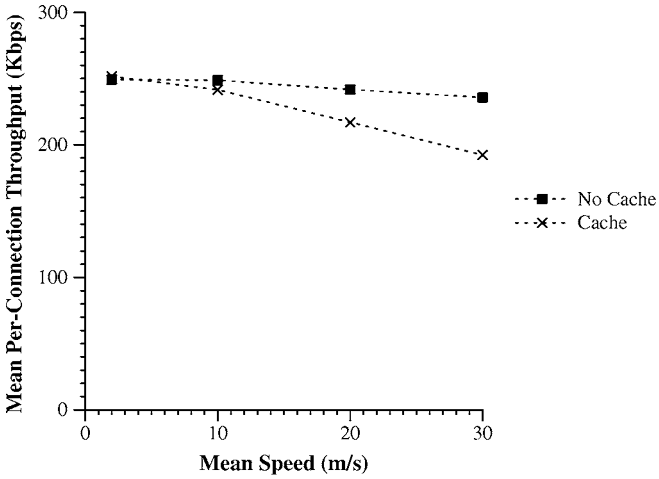



Figure 9. A comparison of TCP-Reno performance of five TCP connections when DSR route replies from cache are, and are not, allowed.

(each between separate pairs of nodes) for 30 patterns. Here, also, we see the same trend as in the previous figure.

Another interesting effect of a routing protocol's behavior with respect to mobility was observed in our second example (section 5.2). The fact that the TCP data flow was lost at the same point in the mobility pattern for both runs raised questions about what characteristic of the pattern was causing the failure. From figure 5(a), it is clear that the rapid sequence of path changes at the 0.13 mark caused all four runs to fail. Upon further inspection, we observed that the routing protocol regularly failed when the minimum path increased in length. This is apparent in the results shown in figure 10.

In the first few moments of the mobility pattern, shown in figure 10(a), the TCP sender and receiver move closer to each other, shortening the path between them from two hops to one (around mark 0.01). A few moments later (around mark 0.07), they slowly diverge to a distance of five hops. In the TCP throughput measurements shown in figure 10(b)–(e), it is evident that the data flow across the TCP connection is main-

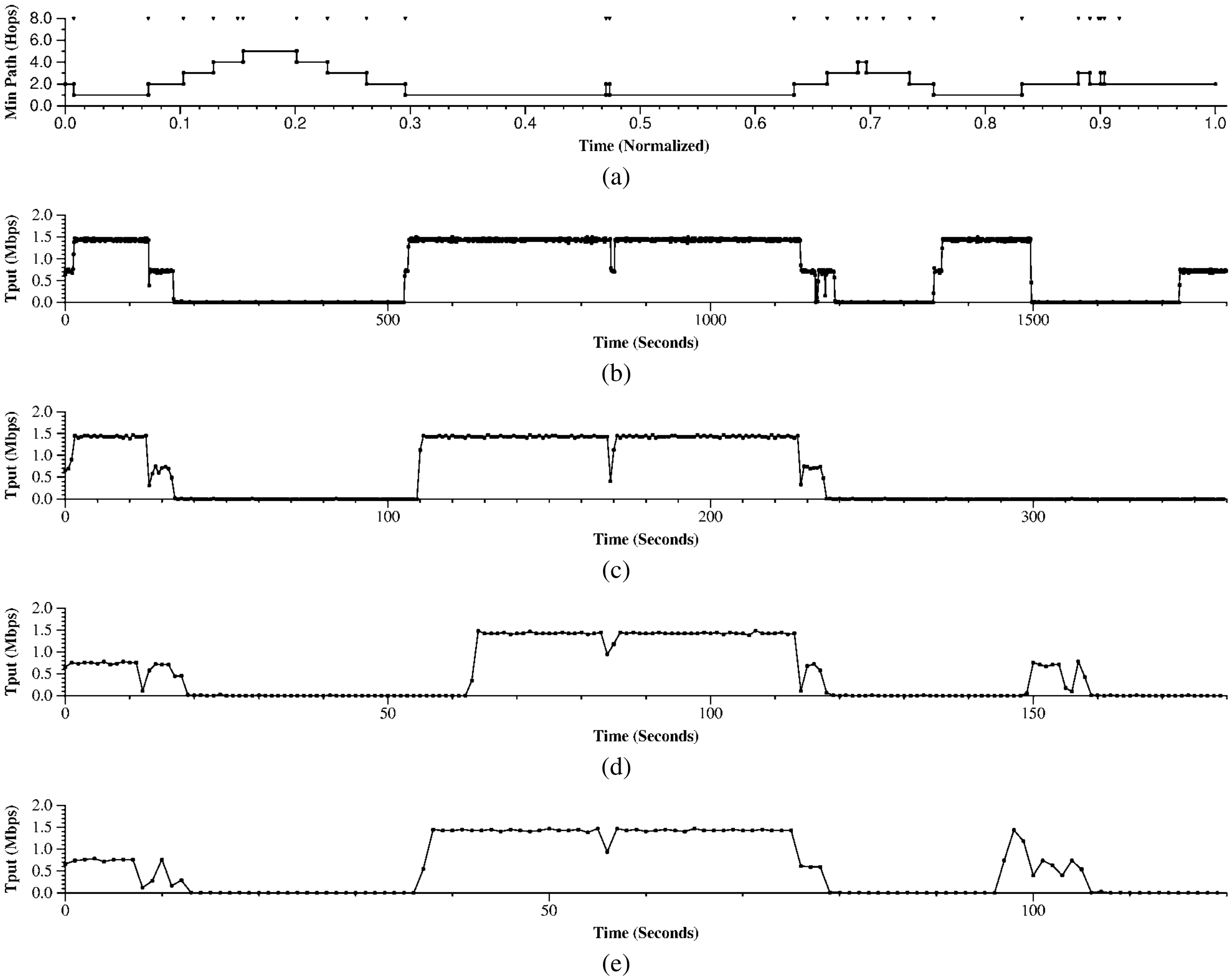



Figure 10. TCP-Reno performance for mobility pattern No. 46, showing that an increase in the minimum path length between the TCP sender and receiver consistently results in the loss of data flow across the connection. (a) Mobility pattern profile. Mean speed (in m/s) is (b) 2, (c) 10, (d) 20, and (e) 30. The ticks at the top of (a) denote changes on the minimum path between the TCP sender and receiver. The curves in (b)–(e) show the measured throughput for the connection, averaged over 1 s intervals.

tained when the path is shortened, but is lost when the path is lengthened. This happens several times in the pattern, independent of the mean speed of the nodes. Most notably, figure 10(b) shows that even while traveling at a slow speed of 2 m/s, a path change from one hop to two (around the 1500 s mark) can stall the data flow. This behavior can be attributed, in part, to the routing protocol. As the TCP sender and receiver move closer to each other, DSR can often maintain a valid route by shortening the existing route, and often does so before a failure occurs. However, as the TCP sender and receiver diverge, the increase in path length eventually causes a route failure because DSR does not attempt to lengthen a route until a failure occurs. The route failure and subsequent route discovery process often result in the restoration of the route only after the TCP sender has repeatedly timed out and backed off, stalling the data flow. This is further magnified by the caching and propagation of stale routes, as mentioned previously.

However, intuition suggests that this is not a problem that is unique to DSR, but will most likely be a problem for other reactive protocols as well. Thus, perhaps a metric of routing protocol performance should not only measure the protocol's ability to recognize optimal routes, but also to quickly adjust an existing route, albeit non-optimally.

Another characteristic of DSR that we observed affecting TCP performance was the route request retransmission backoff algorithm. In DSR, if a route request does not generate a timely reply, the requester times out and retransmits the request. Each timeout results in exponential backoff, which is limited to some fixed maximum value. If this value is too large, then route requests may occur too infrequently to recognize available routes in time to prevent TCP's retransmission timer from backing off to a large value, but if it is too small, then the frequent route requests may cause network congestion. The maximum value suggested in [6] may not be suitable for good TCP performance.

Based on these observations, it might be suggested that instead of augmenting TCP/IP, it would be better to improve the routing protocols so that mobility is more effectively masked. Clearly, extensive modifications to upper layer protocols is less desirable than a routing protocol that can react quickly and efficiently such that TCP is not disturbed. However, regardless of the efficiency and accuracy of the routing protocol, network partitioning and delays will still occur because of mobility, which cannot be hidden.

Thus, in the next section, we analyze some simple modifications to TCP/IP to provide TCP with a mechanism by which it can recognize when mobility induced delays and losses occur, so that it can take appropriate actions to prevent the invocation of congestion control.

## 6. TCP performance using explicit feedback

In this section, we present an analysis of the use of explicit feedback on the performance of TCP in dynamic networks. The use of explicit feedback is not new, and has been proposed as a technique for signaling congestion (e.g., ECN [15], BECN [26], DECBit [28]), corruption due to wireless transmission errors (e.g., EBSN [1], ELN [3]), and link failures due to mobility (e.g., [7], SCPS-TP [9], TCP-F [8]). Our interest in this section is analyzing the performance of the last technique, which we refer to as Explicit Link Failure Notification (ELFN) techniques. Although the TCP-F paper studies a similar idea, the evaluation is not based on an ad hoc network. Instead, they use a black-box, that does not include the evaluation of the routing protocol.

The objective of ELFN is to provide the TCP sender with information about link and route failures so that it can avoid responding to the failures as if congestion occurred.

There are several different ways in which the ELFN message can be implemented. A simple method would be to use a "host unreachable" ICMP message as a notice to the TCP sender. Alternatively, if the routing protocol already sends a route failure message to the sender, then the notice can be piggy-backed on it. This is the approach we took in this analysis. We modified DSR's route failure message to carry a payload similar to the "host unreachable" ICMP message. In particular, it carries pertinent fields from the TCP/IP headers of the packet that instigated the notice, including the sender and receiver addresses and ports, and the TCP sequence number. The addresses are used to identify the connection to which the packet belongs, and the sequence number is provided as a courtesy.

TCP's response to this notice is to disable congestion control mechanisms until the route has been restored. This involves two different issues: what specific actions TCP takes in response to the ELFN notice, and how it determines when the route has been restored.

We used the following simple protocol. When a TCP sender receives an ELFN, it disables its retransmission timers and enters a "standby" mode. While on standby, a packet is sent at periodic intervals to probe the network to see if a route has been established. If an acknowledgment is received, then it leaves standby mode, restores its retransmission timers, and continues as normal. For this study, we elected to use packet probing instead of an explicit notice to signal that a route has been reestablished.

To see what could be achieved with this protocol, we studied variations in the parameters and actions and measured their effects on performance. In particular, we looked at the following:

- Variations in the length of the interval between probe packets.
- Modifications to the retransmission timeout value (RTO) and congestion window upon restoration of the route.
- Different choices of what packet to send as a probe.

The results of these studies are presented below. Each curve is based on the mean throughput for the 50 different mobility patterns we used earlier.

Figure 11 is the analogue of figure 4, except that the results in figure 11 are based on simulations in which TCP-Reno was modified to use ELFN (with a 2 s probe interval). Clearly, the use of ELFN has improved the mean throughput for each of the speeds, as evidenced by the closer proximity of the measured pattern throughputs to the expected throughput line. The tighter clustering of the points also suggests that the use of ELFN techniques improves throughput across all patterns, rather than dramatically increasing just a few. However, notice that for one pattern performance was worse when ELFN was used. In figure 4(c) there is a pattern which has a measured throughput very near to its expected throughput (i.e., it is very close to the line), which is not present in figure 11(c). In this instance, the unusually good performance of TCP was a consequence of fortuitous timing of packet retransmissions, with regard to the state of the network, that did not occur when ELFN was used. This is further evidence of the complex nature of TCP. The general trend, however, shows a performance improvement when ELFN is used.

Figure 12 shows the measured throughput as a percentage of the expected throughput for various probe intervals. Based on these results, it is apparent that the throughput is critically dependent on the time between probe packets. This dependency exists because increasing the time between probes delays the discovery of new routes by the length of the interval. Thus, it is no surprise that if the probe interval is too large, then the throughput will degrade below that of standard TCP, as shown by the results for probe intervals of 30 s. Intuitively, if the probe interval is too small, then the rapid injection of probes into the network will cause congestion and lower throughput. Thus, instead of a fixed interval, perhaps choosing an interval that is a function of the RTT could be a more judicious choice. However, based on the sensitivity of the throughput to the interval size, the function must be chosen very carefully.

In addition to varying the probe intervals, we also looked at the performance advantages of adjusting the congestion window and/or retransmission timeout (RTO) after the failed route had been restored. These results are shown in figure 13. In the figure, ELFN represents the case where no changes are made to TCP's state because of ELFN. Thus, TCP's state (congestion window, RTO, etc.) are the same after the route is restored, as it was when the ELFN was first

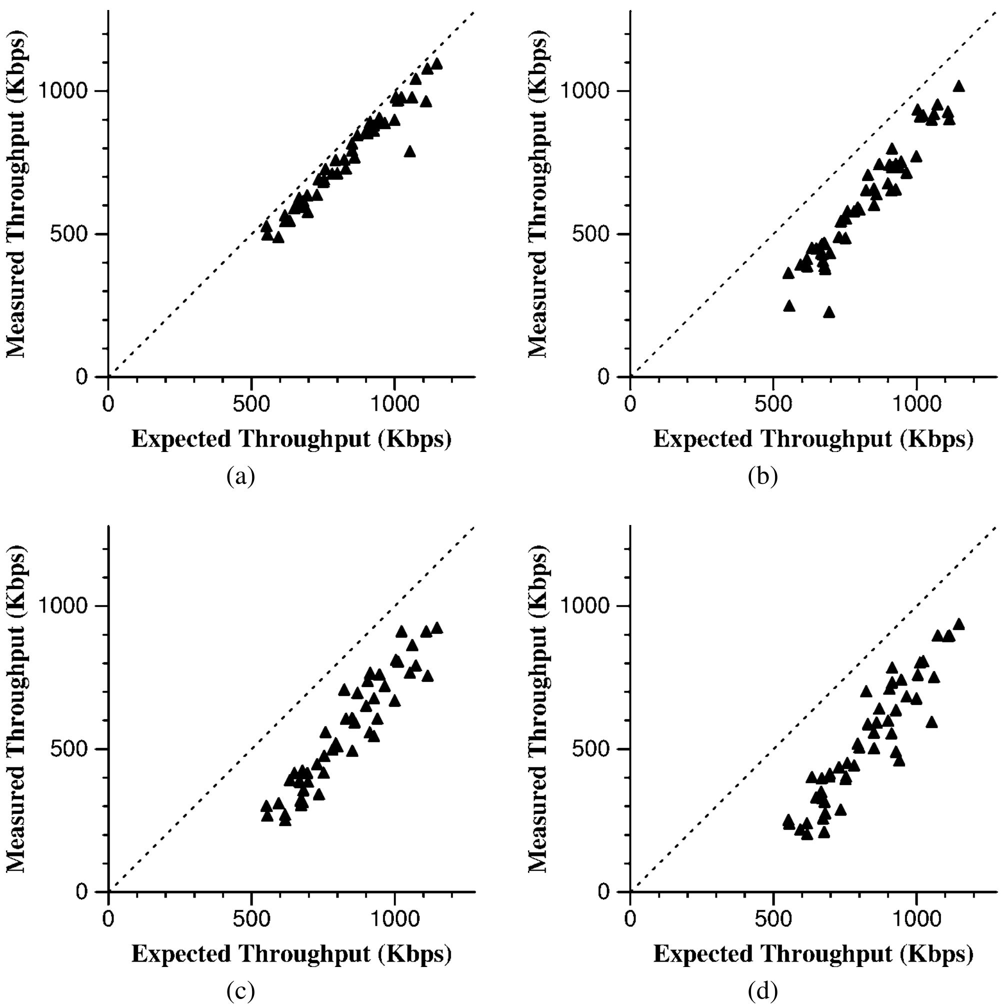



Figure 11. Per-pattern performance of TCP with ELFN using a 2 s probe interval. Speed (in m/s) is (a) 2, (b) 10, (c) 20, and (d) 30.

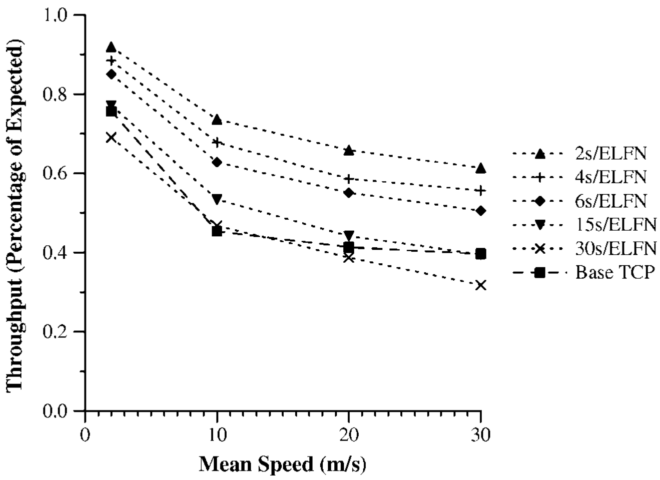



Figure 12. Performance comparison between basic TCP-Reno and TCP-Reno with ELFN using varying probe intervals.

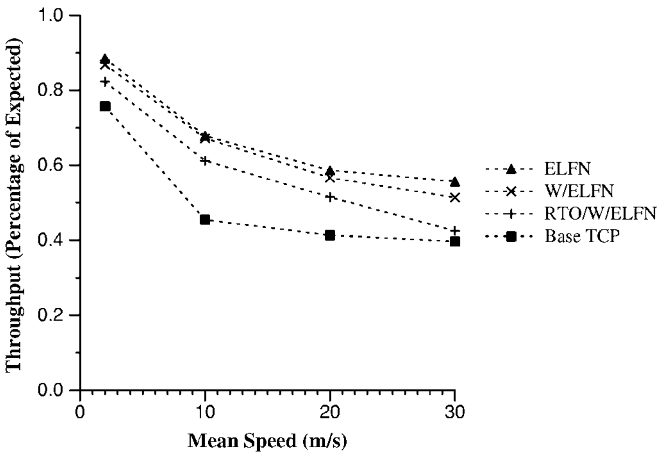



Figure 13. Performance comparison of different window and RTO modifications in response to the receipt of an ELFN message.

received. W/ELFN represents the case where the congestion window is set to one packet after the route has been restored, and RTO/W/ELFN represents the case where the RTO is set to the default initial value (6 s in these simulations) and the window is set to one after the route is restored. Adjusting the window seemed to have little impact on the results. This is believed to be due to the fact that the optimal window (the bandwidth/delay product) of the simulated network is a relatively small number of packets, so it takes only a few round trips to ramp up to the optimal window after a failure. However, altering the RTO had a more significant impact on throughput. We

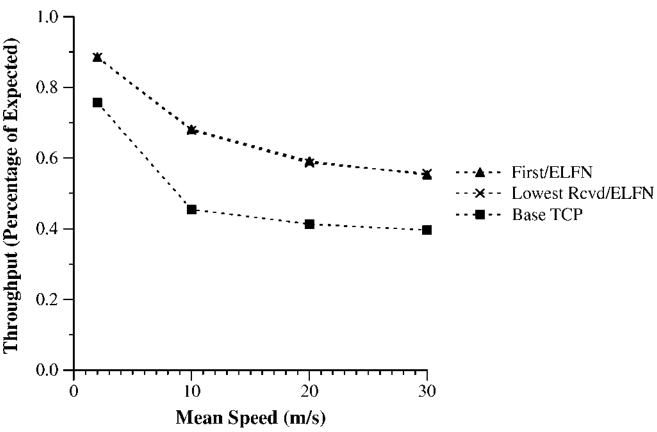



Figure 14. Performance comparison between basic TCP-Reno and TCP-Reno with ELFN using different choices for the probe packet.

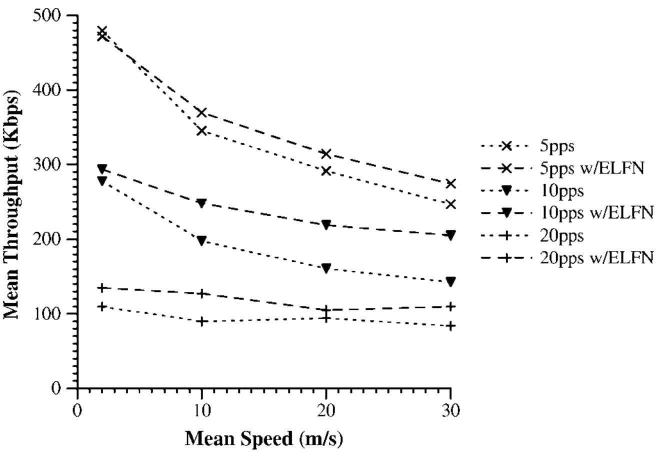



Figure 15. Performance comparison of TCP-Reno and TCP-Reno with ELFN when additional traffic is in the network. The additional traffic is provided by 10 CBR connections, each sending 5, 10, and 20 packets per second (pps).

suspect that this is due to a combination of factors, but is most probably caused by the frequency at which routes break, coupled with ARP's proclivity, as implemented, to silently drop packets. Thus, if a restored route immediately breaks again and results in a failed ARP lookup, then the sender will likely timeout. Given the length of the timeout, it does not take many such occurrences to dramatically affect performance.

We also took a brief look at the impact that the choice of probe packet had on performance, which is shown in figure 14. We considered two possibilities: always send the first packet in the congestion window (First/ELFN in the figure), or retransmit the packet with the lowest sequence number among those signaled as lost in the ELFNs that were received (Lowest Rcvd/ELFN). The first approach is intuitive, the second approach was chosen with the optimistic thinking that perhaps some packets in the window did get through, and, if the route is restored quickly, then the next packet in sequence will be in flight. However, as shown by the results, this had almost no impact whatsoever. We suspect that this has to do with the fact that routes, once broken, were rarely restored quickly. In addition, as shown in section 5, the presence of different forward and reverse routes equalizes the two

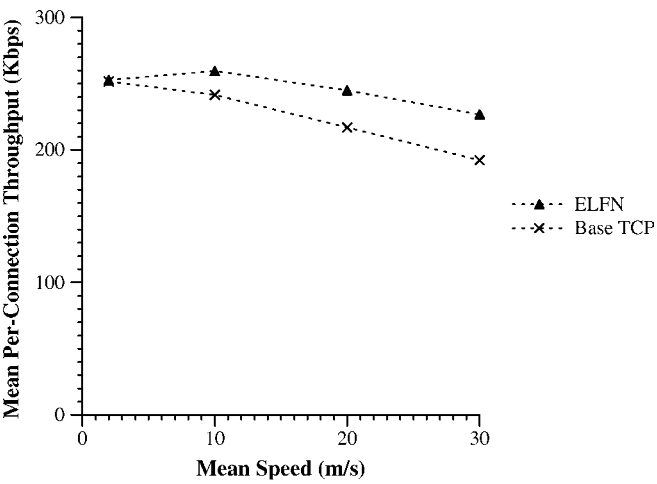



Figure 16. Performance comparison of TCP-Reno and TCP-Reno with ELFN for five concurrent TCP connections.

approaches when only the forward link breaks, since those packets that did get through before the break are acknowledged via the reverse channel. Thus, the lowest sequence number of the packets lost would also happen to be the first in the window.

Finally, we looked at how well the ELFN protocol performs in networks that contain multiple data sources by repeating the set simulations that were used for the throughput curves with cache replies enabled in figures 8 and 9, only now using ELFN as well. The results with CBR traffic are shown in figure 15, and the results for multiple TCP connections are shown in figure 16. Each curve is the average over 30 patterns. The ELFN protocol used 4 s probes. Based on these results, it appears that similar performance benefits can be expected in congested networks, as in the uncongested network.

## 7. Related work

Because routing is an important problem in mobile ad hoc networks, researchers have explored many routing protocols for this environment (e.g., [10,11,13,16,18,20,22,25,27,29–35]), many based on, or developed as a part of, work produced by early DARPA packet-radio programs such as PRNet [21], and SURAN [23,24].

Recently, some researchers have considered the performance of TCP on multi-hop networks [8,17]. Gerla et al. [17] investigated the impact of the MAC protocol on performance of TCP on multi-hop networks. Chandran et al. [8] proposed the TCP-Feedback (TCP-F) protocol, which uses explicit feedback in the form of route failure and reestablishment control packets. Performance measurements were based on a simple one-hop network, in which the link between the sender and receiver failed/recovered according to an exponential model. Also, the routing protocol was not simulated.

Durst et al. [12] looked at the Space Communications Protocol Specifications (SCPS), which are a suite of protocols designed by the Consultative Committee for Space Data Systems (CCSDS) for satellite communications. SCPS-TP handles link failures using explicit feedback in the form of SCPS Control Message Protocol messages to suspend and resume a TCP sender during route failure and recovery. Performance measurements focused on link asymmetry and corruption over last-hop wireless networks, common in satellite communications.

## 8. Conclusions and future work

In this paper, we investigated the effects of mobility on TCP performance in mobile ad hoc networks. Through simulation, we noted that TCP throughput drops significantly when node movement causes link failures, due to TCP's inability to recognize the difference between link failure and congestion. We then made this point clearer by presenting several specific examples, one of which resulted in zero throughput, the other, in an unexpected rise in throughput with an increase in speed. We also introduced a new metric, *expected throughput*, which provides a more accurate means of performance comparison by accounting for the differences in throughput when the number of hops varies. We then used this metric to show how the use of explicit link failure notification (ELFN) can significantly improve TCP performance, and gave a performance comparison of a variety of potential ELFN protocols. In the process, we discovered some surprising effects that route caching can have on TCP performance.

In the future, we intend to investigate ELFN protocols in more detail, as well as the effects that other mobile ad hoc routing protocols have on TCP performance. Currently, we are also studying the impact that the link-layer has on TCP performance, such as aggregate delay caused by local retransmissions over multiple wireless hops.

More research is needed to better understand the complex interactions between TCP and lower layer protocols when used over mobile ad hoc networks, and to find solutions to the problems caused by these interactions. One such problem that we identified was the interaction between TCP and ARP. The ARP in the extensions is based on a BSD implementation, with a one-packet queue and no request timeout mechanism. Thus, packets were regularly dropped or held indefinitely while awaiting resolution. A more advanced ARP needs to be employed, such as one that will provide for the queuing of multiple packets awaiting resolution, with a timeout mechanism to promptly signal failure. Another problem we identified was the significant impact that route cache management has on TCP performance. The results suggest that more aggressive cache management protocols are needed to counter the effects of mobility, such as the use of adaptive route cache timeouts, negative information, or signal strength information.

## Acknowledgements

This work was supported in part by the National Science Foundation under Grant Nos. CDA-9529442 and ANI-9973153, the Texas Advanced Technology Program under Grant No. 010115-248, and the Department of Education under Award No. P200A80305. We would like to thank David Maltz, Josh Broch, and David Johnson at CMU for making their extensions to *ns* available, and for initial support. We would also like to thank the reviewers for their helpful comments.

## References

[1] B.S. Bakshi, P. Krishna, D.K. Pradhan and N.H. Vaidya, Improving performance of TCP over wireless networks, in: *International Conference on Distributed Computing Systems* (May 1997).

[2] H. Balakrishnan and R. Katz, Explicit loss notification and wireless web performance, in: *IEEE Globecom Internet Mini-Conference*, Sydney (October 1998).

[3] H. Balakrishnan, V. Padmanabhan, S. Seshan and R. Katz, A comparison of mechanisms for improving TCP performance over wireless links, in: *ACM SIGCOMM*, Stanford, CA (August 1996).

[4] H. Balakrishnan, V.N. Padmanabhan and R.H. Katz, The effects of asymmetry on TCP performance, in: *Proceedings of the IEEE Mobicom'97*, Budapest, Hungary (September 1997) pp. 77–89.

[5] J. Broch, D.A. Maltz, D.B. Johnson, Y. Hu and J. Jetcheva, A performance comparison of multi-hop wireless ad hoc network routing protocols, in: *ACM/IEEE International Conference on Mobile Computing and Networking* (October 1998) pp. 85–97.

[6] J. Broch, D.B. Johnson and D.A. Maltz, The dynamic source routing protocol for mobile ad hoc networks, Internet Draft, IETF MANET Working Group (December 1998).

[7] R. Cáceres and L. Iftode, Improving the performance of reliable transport protocols in mobile computing environments, IEEE Journal on Selected Areas in Communications 13 (June 1995).

[8] K. Chandran, S. Raghunathan, S. Venkatesan and R. Prakash, A feedback based scheme for improving TCP performance in ad-hoc wireless networks, in: *Proceedings of International Conference on Distributed Computing Systems*, Amsterdam (1998).

[9] Consultative Committee for Space Data Systems (CCSDS), Space Communications Protocol Specifications – Transport Protocol (SCPS-TP) (September 1997).

[10] M.S. Corson and A. Ephremides, A distributed routing algorithm for mobile wireless networks, Wireless Networks 1(1) (1995) 61–81.

[11] B. Das, E. Sivakumar and V. Bhargavan, Routing in ad-hoc networks using a virtual backbone, Manuscript (1997).

[12] R.C. Durst, G.J. Miller and E.J. Travis, TCP extensions for space communications, in: *Proceedings of MOBICOM'96* (1996).

[13] A. Ephremides, J.E. Wieselthier and D.J. Baker, A design concept for reliable mobile radio networks with frequency hopping signaling, Proceedings of the IEEE 75 (January 1987) 56–73.

[14] K. Fall and K. Varadhan, *ns* notes and documentation, LBNL (August 1998) `http://www-mash.cs.berkeley.edu/ns/`

[15] S. Floyd, TCP and explicit congestion notification, ACM Computer Communication Review 24 (October 1994) 10–24.

[16] J. Garcia-Luna-Aceves, A failsafe routing algorithm for multihop packet radio networks, in: *Proceedings of IEEE INFOCOM'86*, Miami, FL (April 1986) pp. 434–443.

[17] M. Gerla, K. Tang and R. Bagrodia, TCP performance in wireless multi-hop networks, in: *Proceedings of IEEE WMCSA'99*, New Orleans, LA (February 1999).

[18] Z.J. Haas and M.R. Pearlman, The zone routing protocol (ZRP) for ad hoc networks, Internet Draft, Mobile Ad Hoc Network (MANET) Working Group, IETF (August 1998).

[19] G. Holland and N. Vaidya, Analysis of TCP performance over mobile ad hoc networks – Part II: Simulation details and results, Technical report TR99-005, Texas A&M University (1999).

[20] D. Johnson, D.A. Maltz and J. Broch, The dynamic source routing protocol for mobile ad hoc networks, Internet Draft, Mobile Ad Hoc Network (MANET) Working Group, IETF (March 1998).

[21] J. Jubin and J. Tornow, The DARPA packet radio network protocols, Proceedings of the IEEE 75, Special Issue on Packet Radio Networks (1987) 21–33.
[22] Y.-B. Ko and N. H. Vaidya, Location-aided routing (LAR) in mobile ad hoc networks, in: *ACM/IEEE Int. Conf. on Mobile Computing and Networking (MobiCom'98)* (October 1998).
[23] G. Lauer, Advanced protocols for the SURAN packet radio network, in: *Proceedings of the SHAPE Packet Radio Symposium* (1989).
[24] G. Lauer, Packet-radio routing, in: *Routing in Communications Networks* (Prentice-Hall, Englewood Cliffs, NJ, 1995).
[25] S. Murthy and J. Garcia-Luna-Aceves, A routing protocol for packet-radio networks, in: *Proc. First ACM International Conference on Mobile Computing and Networking (ACM Mobicom 95)*, Berkeley, CA (November 1995).
[26] P. Newman, Traffic management for ATM local area networks, IEEE Communications Magazine (August 1994) 44–50.
[27] C.E. Perkins and E.M. Royer, Ad hoc on demand distance vector (AODV) routing, Internet Draft, Mobile Ad Hoc Network (MANET) Working Group, IETF (August 1998).
[28] K.K. Ramakrishnan and R. Jain, An explicit binary feedback scheme for congestion avoidance in computer networks with a connectionless network layer, in: *ACM SIGCOMM'88* (August 1988) pp. 565–571.
[29] R. Ramanathan and M. Steenstrup, Hierarchically-organized, multihop mobile wireless networks for Quality-of-Service support, Mobile Networks and Applications 3 (June 1998) 101–119.
[30] S. Ramanathan and M. Steenstrup, A survey of routing techniques for mobile communication networks, Mobile Networks and Applications (1996) 89–104.
[31] N. Shacham, Hierarchical routing in large, dynamic ground radio networks, in: *Proceedings of the Hawaii International Conference on System Sciences (HICSS)*, Honolulu, HI (January 1985) pp. 292–301.
[32] N. Shacham and J. Westcott, Future directions in packet radio architectures and protocols, Proceedings of the IEEE 75, Special Issue on Packet Radio Networks (1987) 83–99.
[33] R. Sivakumar, P. Sinha and V. Bharghavan, Core extraction distributed ad hoc routing (CEDAR) specification, Internet Draft, Mobile Ad Hoc Network (MANET) Working Group, IETF (October 1998).
[34] M. Spohn and J. Garcia-Luna-Aceves, Efficient routing in packet-radio networks using link-sate information, in: *Proc. of the IEEE Wireless Communications and Networking Conference 1999 (WCNC 99)*, New Orleans, LA (September 1999).
[35] C. Toh, A novel distributed routing protocol to support ad-hoc mobile computing, Wireless Personal Communication (January 1997).

**Gavin Holland** is a Ph.D. candidate in computer science at Texas A&M University. He received his B.S. from Abilene Christian University and his M.S. from Texas A&M, both in computer science. He has held positions at Reliable Computer Technology, the Texas Transportation Institute, and the Texas A&M Laboratory for Computing and Digital Systems Research; including acting as P.I. for NFS and AFOSR funded SBIR projects on his research. His current research is in the area of mobile wireless networking, with past research in the areas of fault tolerance and parallel processing. He is a former DOEd Fellow, and has interned at Microsoft Research.
E-mail: gholland@cs.tamu.edu

**Nitin Vaidya** received his Ph.D. from the University of Massachusetts at Amherst. He is presently an Associate Professor in the Department of Electrical and Computer Engineering at the University of Illinois at Urbana-Champaign. He has held visiting positions at Microsoft Research, Sun Microsystems, and the Indian Institute of Technology-Bombay. His current research is in the areas of wireless networking and mobile computing, and he presently leads several funded research projects in these areas. His research has been funded by various agencies, including the National Science Foundation, DARPA, Microsoft Research and Sun Microsystems. Nitin Vaidya is a recipient of a CAREER award from the National Science Foundation. Dr. Vaidya has served on the program committees of several conferences and workshops, and presently serves as the General Chair for the 2001 ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc). He is presently a speaker for the Distinguished Visitor Program of the IEEE Computer Society. He is a senior member of IEEE Computer Society and a member of the ACM.
E-mail: nhv@crhc.uiuc.edu
WWW: http://www.crhc.uiuc.edu/~nhv/