Analysis of TCP Performance over Mobile Ad Hoc Networks*

Part I: Problem Discussion and Analysis of Results

Gavin Holland and Nitin Vaidya
{gholland,vaidya}@cs.tamu.edu

Dept. of Computer Science
Texas A&M University, College Station, TX 77843
Phone: (409)845-5534 Fax: (409)847-8758

February 15, 1999

Technical Report 99-004

Abstract

Mobile ad hoc networks have gained a lot of attention lately as a means of providing continuous network connectivity to mobile computing devices regardless of physical location. Recently, a large amount of research has focused on the routing protocols needed in such an environment. In this two-part report, 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 because of 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. In this paper (Part I of the report), we present the problem and an analysis of our simulation results. In Part II of this report, we present the simulation and results in detail.

Keywords: Mobile Ad Hoc Networks, TCP/IP, Performance Analysis, Explicit Link Failure Notification (ELFN), Dynamic Source Routing (DSR)

*This work was supported in part by the Department of Education under Award No. P200A80305, the National Science Foundation under Grant No. 32525-46600, and the Texas Advanced Technology Program under Grant No. 010115-248.

1 Introduction

With the proliferation of mobile computing devices, the demand for continuous network connectivity regardless of physical location has spawned an 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 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. [9, 10, 15, 17, 18, 19, 20, 21]. Our research is focused on the performance of TCP over mobile ad hoc networks.

Since TCP/IP is the standard network protocol stack for communication on the Internet, its use over mobile ad hoc networks is a certainty because of the number of applications that it leverages, and because it allows seamless integration with the fixed infrastructure, where available.

However, earlier research on TCP over cellular wireless systems has shown that TCP suffers poor performance because of packet losses and corruption caused by wireless induced errors. Thus, a lot of research has focused on mechanisms to improve TCP performance in cellular wireless systems (e.g. [1, 2]). Other studies have looked at the problem of bandwidth asymmetry and large round-trip times, prevalent in satellite networks(e.g. [11, 3]).

In this report, we address another characteristic of mobile ad hoc networks that impacts TCP performance: link failures due to mobility. In this paper, Part I of the report, we present a performance analysis of standard TCP over mobile ad hoc networks, and then we present an analysis of the use of explicit notification techniques to counter the affects of link failures. In Part II of this report [16], we present details of the simulation environment and comprehensive results for each simulation run.

2 Simulation Environment and Methodology

The results in this report are based on simulations using the *ns* network simulator from Lawrence Berkeley National Laboratory (LBNL) [12] with extensions from the MONARCH project at Carnegie Mellon [4]. 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. We refer the reader to [4] for more information on the extensions. Unless otherwise noted, no modifications were made to the simulator described in [4] beyond minor bug fixes that were necessary to complete the study.

All results are based on a network configuration consisting of TCP-Reno (without delayed acknowledgments) over IP, communicating over an 802.11 wireless network, with routing provided by the Dynamic Source Routing (DSR) protocol and the implementation of BSD's ARP protocol (used to resolve node addresses to MAC addresses among neighboring nodes).

The choice of DSR as the routing protocol was primarily based on the availability of the ns extensions at the time when this study was initiated. Our goal was only to observe TCP's performance in the presence of mobility induced failures in a plausible network environment for which any of the proposed mobile wireless ad-hoc routing protocols would have sufficed. However, since we frequently refer to the routing protocol in this paper, the next paragraph is a brief primer on DSR to familiarize the reader with its terminology and characteristics.

The Dynamic Source Routing (DSR) protocol is a routing protocol for mobile ad-hoc networks developed by researchers at CMU [5]. 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 by the source node through *route discovery*. When a node has a packet 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 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 like snooping routes in route replies or data packets or eavesdropping on local broadcasts. This cache is updated through route error messages that are sent when a packet cannot be delivered because its route is invalid. The route discovery protocol as implemented in the CMU extensions to *ns* actually 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, then 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 for it within a prespecified amount of time (the "Send Buffer Timeout" interval, which is 30s by default) or if the queue is full and new outgoing packets have arrived. Route discoveries for the same destination are limited by the backoff and retry procedure, which is initiated per destination and not per packet. Thus, regardless of the number of packets waiting for 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. In this instance, DSR uses a mechanism called *packet salvaging* to re-route the packet. When a node x detects that the next link in a packet's route is broken, it sends a route error message to the node that generated the packet's route to prevent it from sending more packets on the 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 [4] with a few exceptions.

The following is a discussion of our simulation setup.

Our network model consists of 30 nodes moving around in a 1500x300 meter flat rectangular area, using the *random waypoint* mobility model. In the random waypoint model, each node x picks a random speed and destination in the rectangular area and then travels to the destination in a straight line at the chosen speed. Once node x arrives at its destination it picks another destination and continues onward. So, 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, with a bandwidth of 2Mbps and a transmission radius of 250m.

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 the speed 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's 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, but at a proportionally different rate. More details about the simulation setup are given in Part II of this report [16].

3 Performance Metric

In this performance study, we set up a single TCP connection between a chosen pair of sender and receiver nodes and measured the throughput over the lifetime of the connection. The throughput is used 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. Thus, when a link on a TCP route breaks, the TCP sender may timeout and reduce its congestion window and/or back-off the retransmission timer. Therefore, route changes due to host mobility have a detrimental impact on TCP performance.

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

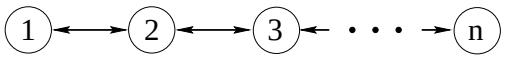
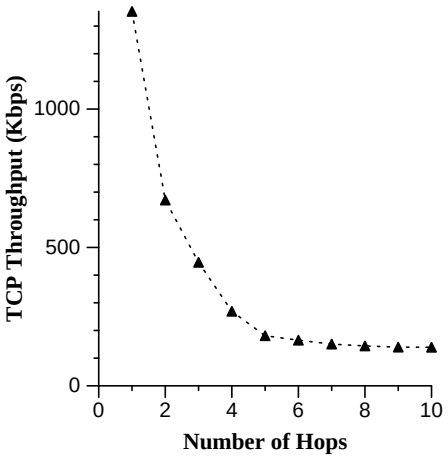


Figure 1: Network topology used for the fixed multi-hop simulation.

We now describe how the *expected throughput* is obtained. We first simulated a *static* network of n nodes such that the n nodes form a linear chain containing $n-1$ wireless hops, as shown in Figure 1 (the topology is fixed). A one-way TCP data transfer is performed between the two nodes at the two ends of the linear



Hops	*Throughput (Kbps)*
1	1352.9
2	671.1
3	446.1
4	269.3
5	181.0
6	164.8
7	150.7
8	144.1
9	139.7
10	139.2

Figure 2: TCP throughput for an 802.11 fixed multi-hop network of varying length.

chain and TCP throughput is measured between these nodes. The nodes use the 802.11 MAC protocol for medium access. This set of TCP throughput measurements is analogous to that performed by Gerla et al. [14], using similar (but not identical) MAC protocols.

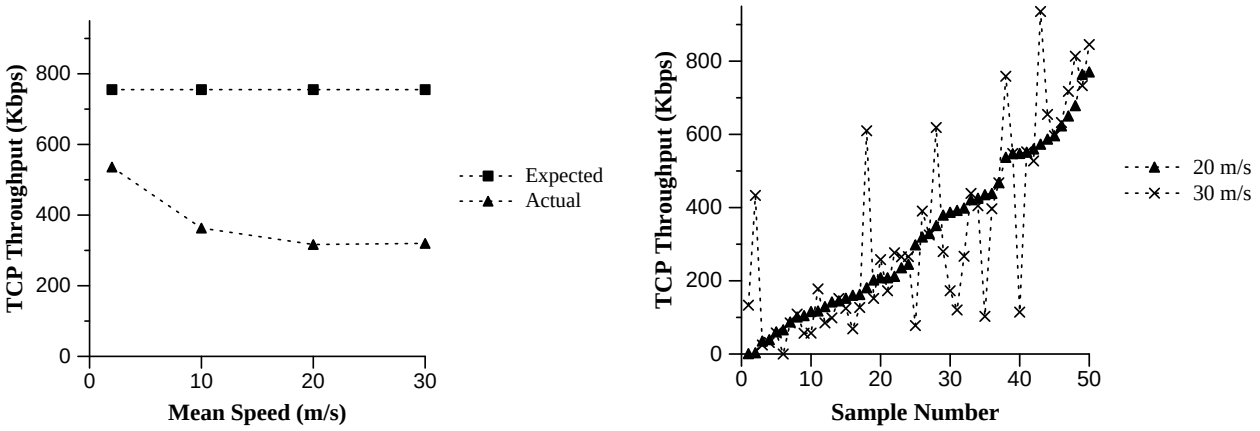
Figure 2 presents the measured throughput as a function of the number of hops. Observe that the throughput decreases rapidly when the number of hops is increased from 1 and then stabilizes once the number of hops becomes large. This trend is similar to that reported in [14]. Therefore, we refer the reader to [14] for a detailed explanation of the reasons behind this trend. Our objective is to use these measurements to determine the *expected throughput*.

The *expected throughput* is a function of the actual mobility pattern. For instance, if two nodes are always adjacent and move together (similar to two passengers in a car), then the expected throughput for the TCP connection between them would be identical to that for 1 hop in Figure 2. On the other hand, if the two nodes are always in different partitions of the network, then 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 destination contains i hops ($1 \leq i \leq \infty$). Let T_i denote the throughput obtained over a linear chain (as in Figure 2) using i hops. When the two nodes are partitioned, we consider that the number of hops i is ∞ and $T_\infty = 0$. Now, the expected throughput is calculated as

$$\text{expected throughput} = \frac{\sum_{i=1}^{\infty} t_i * T_i}{\sum_{i=1}^{\infty} t_i} \tag{1}$$

Of course, $\sum_{i=1}^{\infty} t_i$ is equal to the duration for which the TCP connection is in existence. The actual 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 destination. Also, the above formulation of expected throughput 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 actual 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



(a) Comparison of the measured throughput to the expected throughput.

(b) Comparison of the per-sample measured throughputs for the 20 and 30 m/s points.

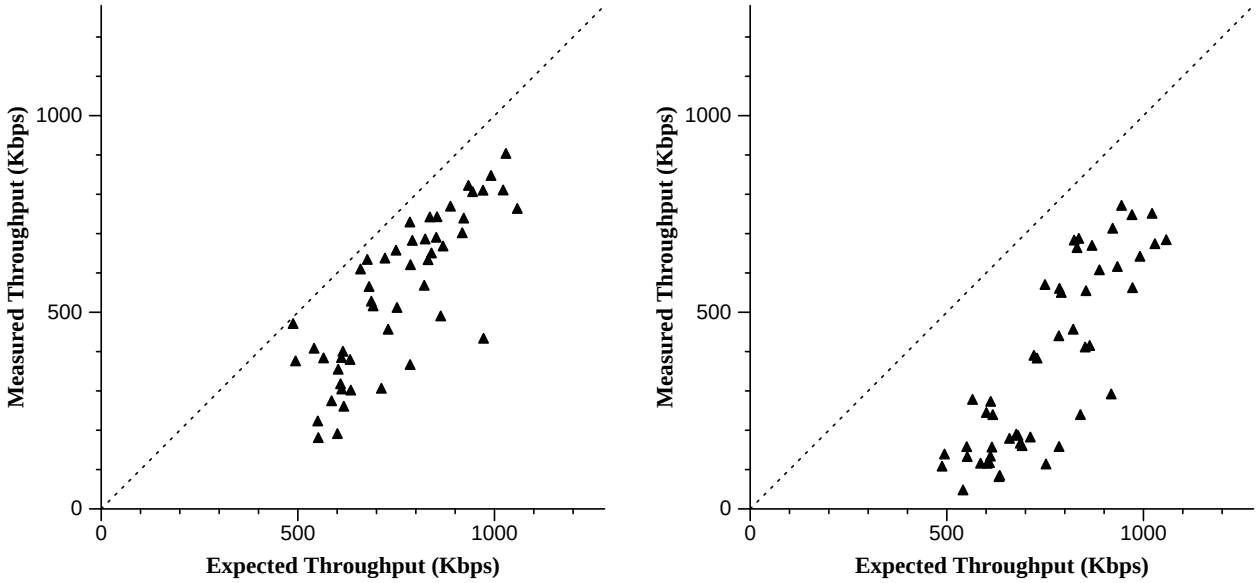
Figure 3: TCP-Reno throughput for a single connection over a mobile ad hoc network.

Figure 3(a) reports the actual throughput of TCP-Reno (averaged over 50 runs), as well as the expected throughput (also averaged over 50 runs), 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, however, the 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.

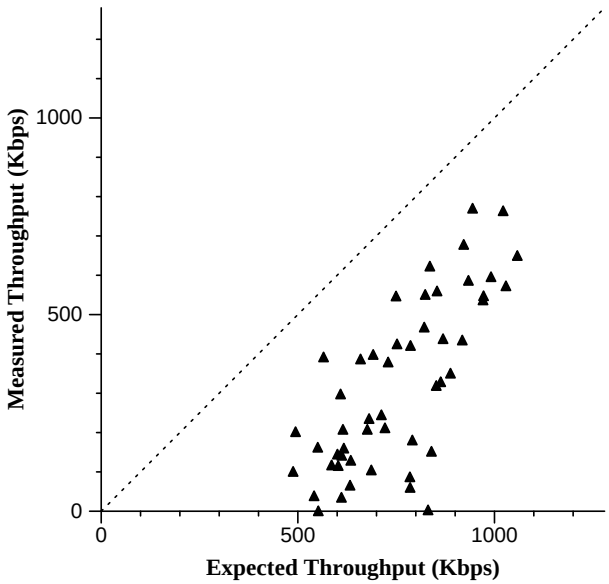
Intuition suggests that when the speed is increased, then route failures happen more quickly, resulting in packet losses and frequent route discoveries. Thus, intuitively, TCP throughput should monotonically degrade as the speed is increased. In Figure 3(a), the throughput drops sharply as the mean speed is increased from 2 m/s to 10 m/s, and to 20 m/s. However, when the mean speed is increased from 20 m/s to 30 m/s, the throughput averaged over the 50 runs slightly increases. This is a counter-intuitive result. To explain this, Figure 3(b) 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 the order of their throughputs at 20 m/s). Observe that, for certain mobility patterns, the throughput increases when the speed is increased. Later, in Section 5, we explain this anomaly.

Figure 4 provides a different view of the TCP throughput measurements. In this figure, we plot the actual throughput versus expected throughput for each of the 50 mobility patterns. The four graphs correspond to four different average 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 actual throughput is closer to the expected throughput, the corresponding point in the graph would be closer to the diagonal line, and vice versa. The following observations can be made from Figure 4:

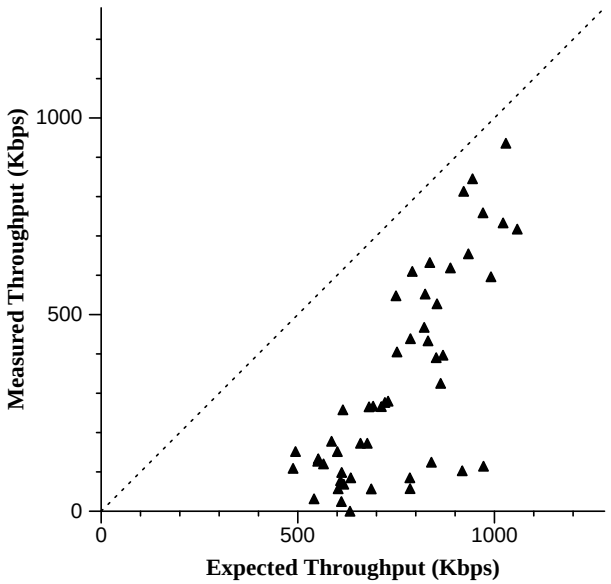


(a) mean speed = 2 m/s

(b) mean speed = 10 m/s



(c) mean speed = 20 m/s



(d) mean speed = 30 m/s

Figure 4: Comparison of actual and expected throughput for 50 mobility patterns.

- 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 will 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 zero 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, then the link between them will not break, independent of their speed).

Section 5 attempts to provide 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, 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 report 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* [1]. In this example, no acknowledgments are received by the TCP source during the duration of the TCP connection although the *expected throughput* for the mobility pattern under consideration is 632 Kbps.

In this scenario, the TCP source and the sink nodes are initially six hops apart, as shown in Figure 5, and stay within six hops of each other for all but 6 seconds of the 120 second simulation. For those 6 seconds, the network is partitioned such that the source and sink nodes are in different partitions. The time they spend in different partitions is shown in Figure 5 as two intervals, one at the beginning and one at the end, in which the distance in hops is zero (which means no possible path exists between the TCP source and sink).

A condensed version of the simulation packet trace for this scenario is shown in Figure 6. This trace is obtained with node 1 being the TCP source and node 2 being the TCP sink. 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

[1] We measure *throughput* as the amount of data that has been *acknowledged* to the sender.

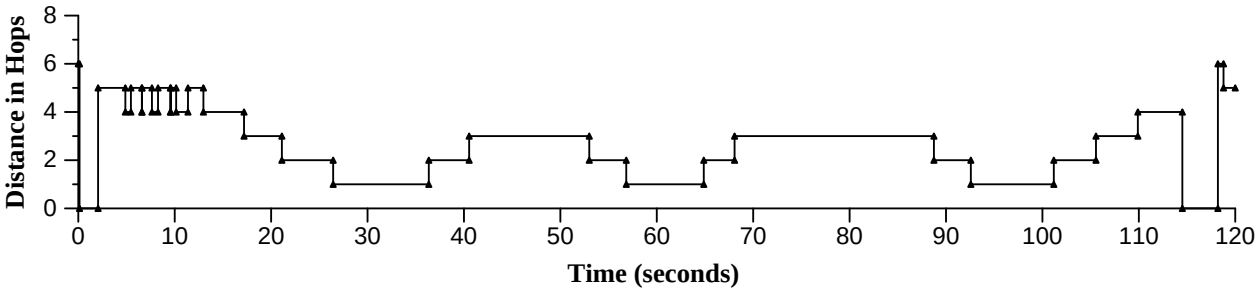


Figure 5: The distance in hops between the TCP source and sink for the duration of the run in the zero-throughput example. The average node speed is 30 m/s.

Evnt	*Time (secs)*	*Node*	*SeqNo*	*Pkt*	*Resn*
s	0.000	1	1	tcp	
D	0.207	5	1	tcp	NRTE
s	6.000	1	1	tcp	
r	6.044	2	1	tcp	
s	6.044	2	1	ack	
D	6.119	21	1	ack	NRTE
s	18.000	1	1	tcp	
D	18.363	27	1	tcp	NRTE
s	42.000	1	1	tcp	
s	90.000	1	1	tcp	
D	90.006	16	1	tcp	ARP
D	120.000	16	1	tcp	END

Figure 6: Packet trace for a 30 m/s run that experienced zero throughput.

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, *ARP* means the ARP protocol failed to locate a MAC address, and *END* means the simulation finished. The *Node*, *SeqNo*, and *Pkt* columns report the node at which the event occured, the TCP sequence number of the packet depicted in the event, and the type of packet, respectively.

Soon after the first packet is sent by the source, a link break occurs along the TCP route that causes a partition in the network (this is the first 0-hop interval shown in Figure 5). The partition causes the first packet to be dropped by the routing protocol (at time 0.207 seconds). Eventually, the TCP sender on node 1 times-out and retransmits the first packet (at time 6.000). On the second attempt, the packet does reach the destination, node 2, which immediately sends an acknowledgment. However, the ack is sent on a "stale" cached route that does not exist anymore (i.e., some links on the route are broken), so, the acknowledgment is also dropped. The remaining attempts to retransmit the packet also fail due to stale cached routes (see the rows with $Evnt = \text{D}$ in Figure 6).

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 2.5 when the speed is increased from 10 m/s to 20 m/s. In the scenario under consideration, the TCP source and sink were able to reach each other 95.5% of the time and spent 64% of the time at most two hops away. Except for the short durations of time when the nodes were in different partitions of the network, the nodes were never more than five hops away.

The characteristics of the TCP connection between the source and sink are shown in Figure 7, which shows the distance in hops between the source and sink for the duration of the connection. The x-axis is shown as normalized time to reflect the fact that the pattern is constant regardless of node speed, as mentioned in Section 2.

As shown in Figure 7, during the first quarter of the duration of the TCP connection the distance between the source and sink nodes initially fluctuates rapidly between four and five hops and then slowly converges to one hop for the remainder of the quarter. This is followed, in the second quarter, by a gradual separation and then a fluctuation around three and four hops, including a brief interval of time in which the network is partitioned (around the 0.32 mark). For the last half of the duration, the nodes are one to two hops away.

Figure 8 shows the data packets sent by the TCP source for each run, where graphs (a) and (b) show the results for runs with mean node speeds of 10 m/s and 20 m/s, respectively. As mentioned earlier, the sequence of moves that each node makes is identical in the 10 m/s run in graph (a), as it is in the 20 m/s run in graph (b). The only difference is that a distance covered by a node, say x, over time t in (a) takes x a time of $t/2$ to cover in (b). 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 $2r$. In this instance, the sequence of frames is the mobility pattern shown in Figure 7.

Discussion of Figure 8(a) In the 10 m/s run, the routing protocol was able to maintain forward and reverse routes during the initial instability, resulting in good initial throughput. The variations in the throughput (shown as variations in the rate at which packets are sent) are due to the distance in hops between the nodes. However, the gradual change in distance from two hops to three hops around the 95s mark results in TCP backoff from which the connection never recovers. The details of the packet activity at the moment at which the initial backoff occurs is shown in Figures 9(a) and (b). Leading up to the loss (at the 95.3s mark), the forward and reverse routes were different. Around the 95.3s mark, the forward route breaks at the link between the source and the next hop in the route, due to mobility. Since the backward route still exists, all of the outstanding acks are delivered, triggering the queuing of a full window at the source. This appears around the 95.4s mark as a burst of packets. In response to the route failure, the routing protocol on the source finds an alternate four hop route in its cache and delivers the full window to the next node in the new route. However, because the route is stale, the third node in the route drops half of the packets around the 95.7s mark after a failed attempt at salvaging them, and then sends a route failure

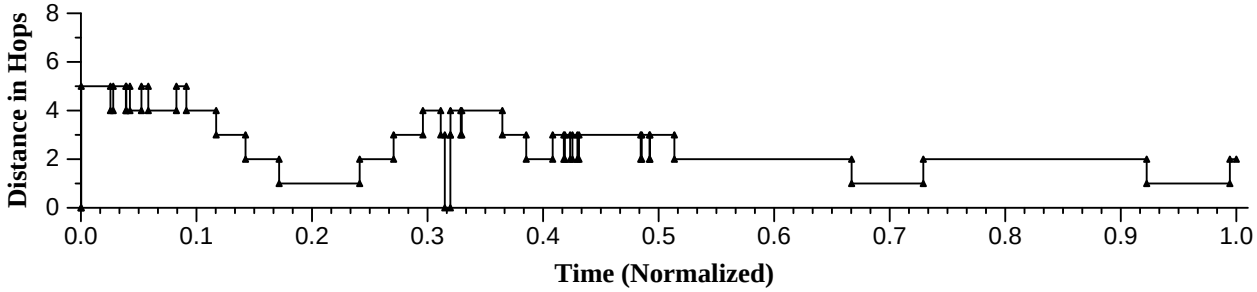
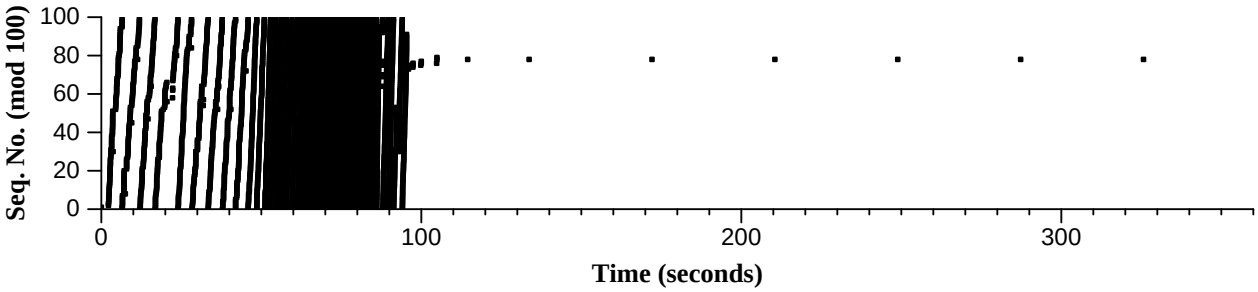
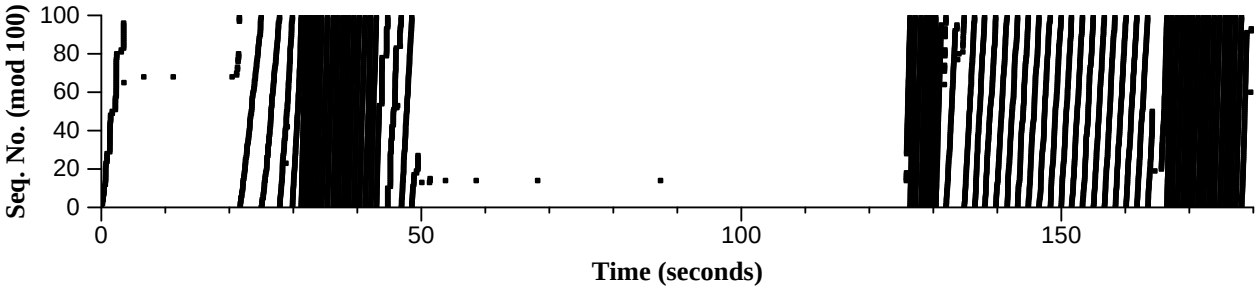


Figure 7: The distance in hops between the TCP source and sink for the mobility pattern in the anomalous-throughput example. The x-axis is normalized to reflect the fact that the pattern is constant regardless of node speed (and, therefore, simulation duration).



(a) mean speed = 10 m/s



(b) mean speed = 20 m/s

Figure 8: The packets sent for the same pattern, but with different mean node speeds, showing that throughput actually increases with an increase in speed.

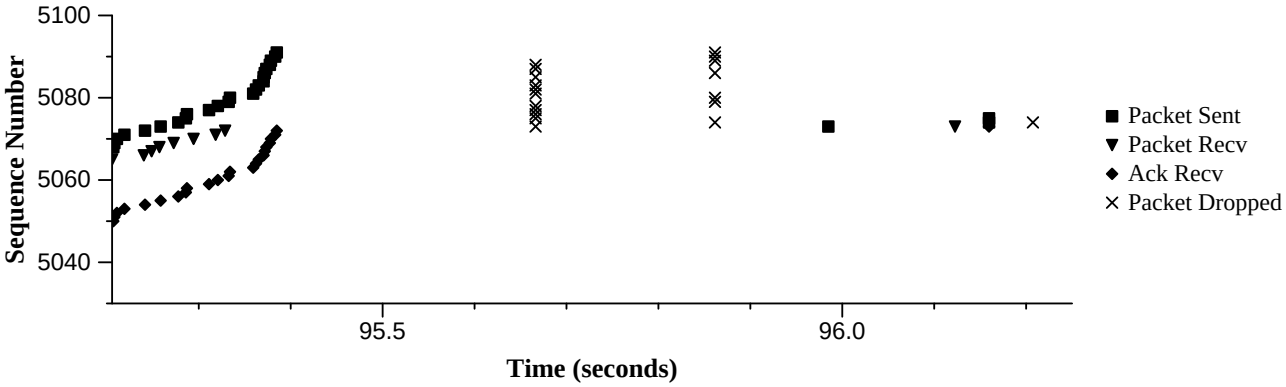


Figure 9: Detailed packet plot showing the moment at which TCP enters repeated backoff in the 10m/s run. *Packet Sent* and *Packet Recv* indicates the time at which a packet with the indicated sequence number was sent by the source and arrived at the destination, respectively, *Ack Recv* indicates the time at which an acknowledgment was received by the sender with the indicated sequence number, and *Packet Dropped* indicates the time at which the packet with the indicated sequence number was dropped.

packet to the source (not shown). It later drops the other half of the packets that arrive around the 95.9s mark, which were buffered in the previous node's interface. At the 96.0s mark, the TCP source times-out and retransmits on another route it finds in its cache. This time, however, the packet and its ack make a successful round-trip after both are delayed by route salvaging attempts at intermediate nodes. Due to the salvage attempts, the source searches for a new route for the next packet through route-discovery after exhausting its own route cache. It chooses one of the replies it receives and sends out the next packet along the new route. However, this is, again, quickly dropped because the new route is now stale, even though, at this point, the source and sink are still only two hops away. Similarly, for all subsequent timeouts except one, stale routes result in packet loss even though the source and sink are never more than four hops distance from each other. The one exception occurs around the 114s mark, at which a retransmitted packet is sent within the small 1.7s window during which the network is partitioned, and lost.

Discussion of Figure 8(b) The 20 m/s run shares many of the characteristics of the slower 10 m/s run, but it results in higher throughput because a retransmission late in the pattern (around the 125s mark in Figure 8(b)) succeeds in re-establishing the flow of packets in spite of poorer performance at the beginning of the pattern. Initially, the routing protocol was unable to maintain a route during the initial instability, resulting in TCP backoff, as seen by the gap in the first 20 seconds of Figure 8(b). However, a retransmission around the 20s mark results in valid forward and reverse routes after several salvage attempts. The throughput, again, degrades when repeated route failures induce packet losses, causing the TCP source to timeout and backoff. This loss occurs at the same point in the pattern with 10 m/s and 20 m/s speeds (the 100s mark in Figure 8(a) and the 50s mark in Figure 8(b)); at this point, the nodes are gradually moving apart. However, unlike the 10 m/s run, the packet flow is re-established later in the pattern (at the 125s mark) when a route is found for a retransmitted packet after the nodes have converged to within one hop of each other. This success is why the second run has twice the throughput of the first run.

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 stale route caching. Even in relatively slowly changing topologies, the inability of the routing protocol to recognize stale routes resulted in repeated route failures. Furthermore, allowing the intermediate nodes to reply to a route request with routes from their cache further delayed route discovery because it prevented the propagation of the route request to the destination. This was because the intermediate nodes frequently returned stale routes. However, we believe that this problem can potentially be solved by tweaking the route cache timeout, perhaps dynamically, depending on a node's observed route failure rate. Alternatively, replying from caches can be

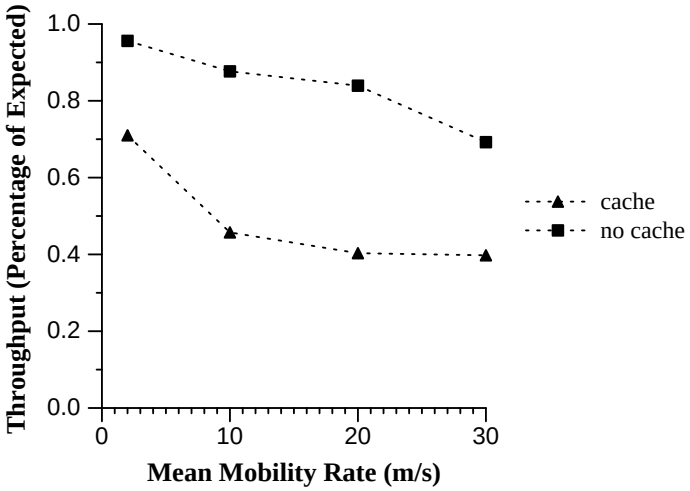


Figure 10: A comparison of TCP performance with and without route replies from caches.

turned off altogether. This has a startling improvement in performance, as shown in Figure 10. However, these results are for a single TCP connection in an uncongested network. In a network with multiple connections, the additional routing traffic introduced when caching is not used should significantly degrade TCP performance.

Another interesting effect of a routing protocol's behavior with respect to mobility was observed in our second example (Figure 8). The fact that both runs failed at the same point in the mobility pattern raised questions about what characteristic of the pattern was causing difficulties for the routing protocol. Upon inspection, we learned that, at the point of failure, the TCP source and sink nodes are passing by each other in opposite directions, in a crossing pattern. As they approach, the routing protocol is able to easily maintain a route by shortening the existing route. However, after they cross and diverge, the routing protocol fails to successfully lengthen the route. This is because this implementation of DSR relies on salvaging to append the last hop until a new route can be found. Unfortunately, because of the caching of stale routes, the

salvaging fails to deliver the stranded packets, and the stale routes returned by the source's neighbors delays its ability to find a valid route until TCP has already repeatedly timed-out. 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 its ability to recognize optimal routes, but also to quickly adjust an existing route, albeit non-optimally.

Another problem we observed was delays caused by large backoff during the retransmission of route requests. In DSR, if a route request does not generate a reply, then the requester times-out and retransmits the request. Each timeout results in exponential backoff, with a fixed maximum value. If this value is too large, then route requests occur too infrequently to recognize available routes in time to prevent TCP's retransmission timer from backing-off to a large value. However, there is an obvious tradeoff between the advantages of rapid route discovery and the extra congestion induced by the propagation of frequent route requests that must be studied carefully before a suitable value is determined.

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 undesirable, and if a routing protocol is found that can react quickly and efficiently enough such that TCP is not disturbed, this would be a desirable solution. However, regardless of the efficiency and accuracy of the routing protocol, network partitioning and delays will still occur, 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 an appropriate action 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(ECN [13]), corruption due to wireless transmission errors(ELN [1, 2]), and link failures due to mobility ([6], SCPS-TP [8], TCP-F [7]). Our interest in this section is analyzing the performance of the latter, 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 occured.

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 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 this message. 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 source and destination 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 to the TCP sender.

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, and how TCP determines that the route has been restored.

We used the following simple protocol. When a TCP source receives an ELFN, it disables its retransmission timers and enters a "stand-by" mode. While on stand-by, a packet is sent at periodic intervals to probe the network to see if a route has been established. If an ack is received, then it leaves the stand-by 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 re-established.

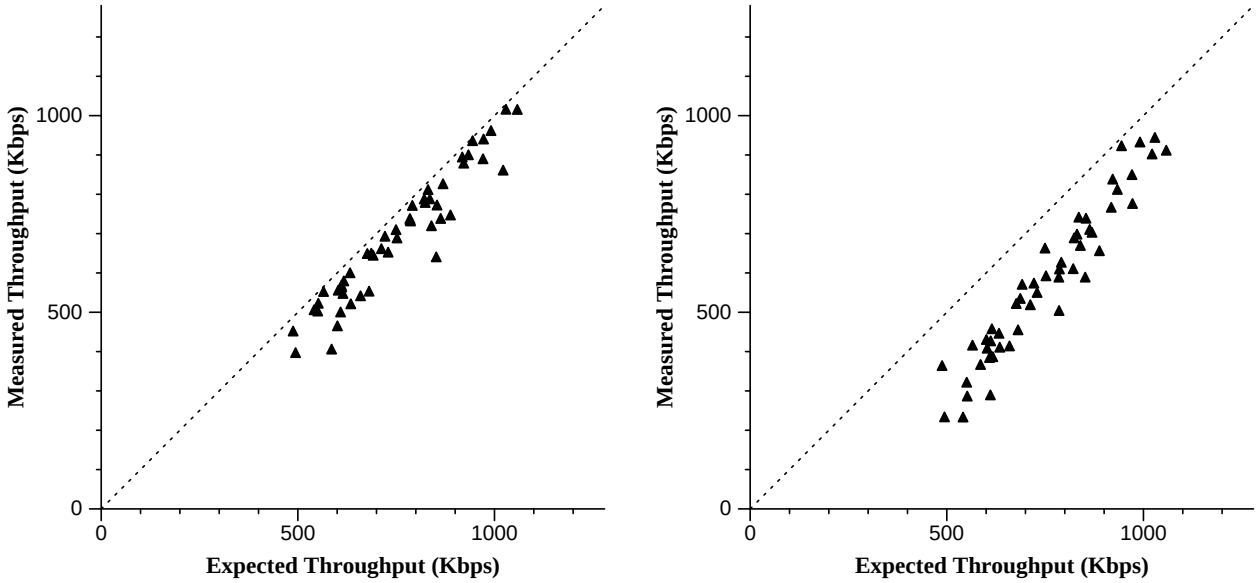
To see what could be achieved with this protocol, we studied variations in the parameters and actions and measured their effects on performance. In particular, we looked at the following:

- Variations in the length of the interval between probe packets.
- Modifications to the 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. Unless otherwise stated, each curve is based on the mean throughput for the 50 different mobility patterns we used earlier.

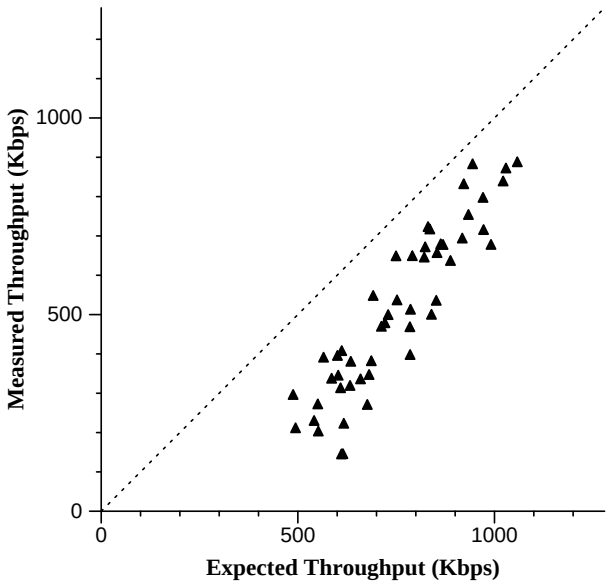
Figure 11 is the analogue of Figure 4, except that the results in Figure 11 are based on simulations in which ELFN is used with a 2s probe interval. Clearly, the use of ELFN has improved the throughput for each of the speeds, as evidenced by the proximity of the measured pattern throughputs to the expected throughput line. The tighter clustering of the points also suggests that the use of ELFN techniques also improves throughput across all patterns rather than dramatically increasing a few.

Figure 12 shows the throughput as a percentage of the expected throughput for varying probe intervals. Based on these results, it is apparent that the throughput is critically dependent on the time between probe packets. This is 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 30s. Intuitively, if the probe interval is too small, then the rapid injection of probes into the network will cause congestion and lower throughput as well. 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, this function must be chosen very carefully.

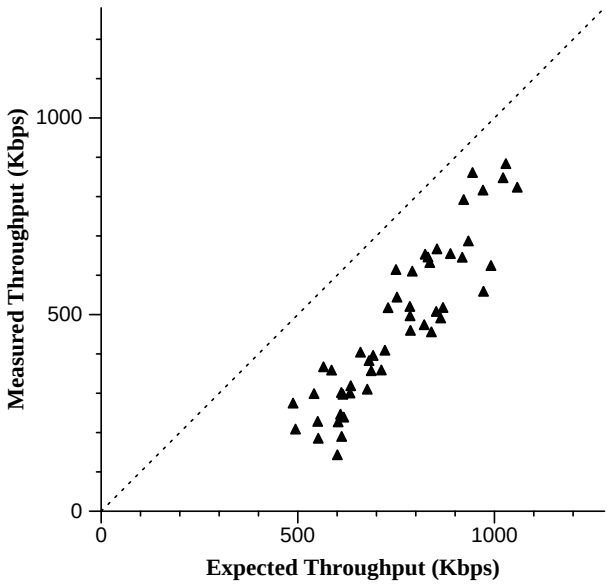


(a) mean speed = 2 m/s

(b) mean speed = 10 m/s



(c) mean speed = 20 m/s



(d) mean speed = 30 m/s

Figure 11: Per-pattern performance of ELFN with a 2s probe interval.

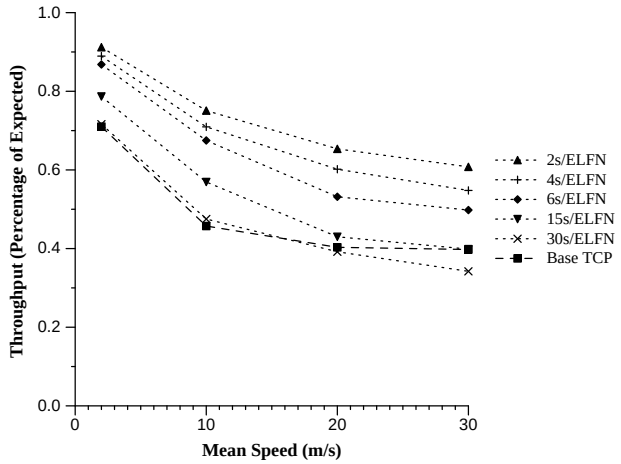


Figure 12: Performance comparison of varying probe intervals.

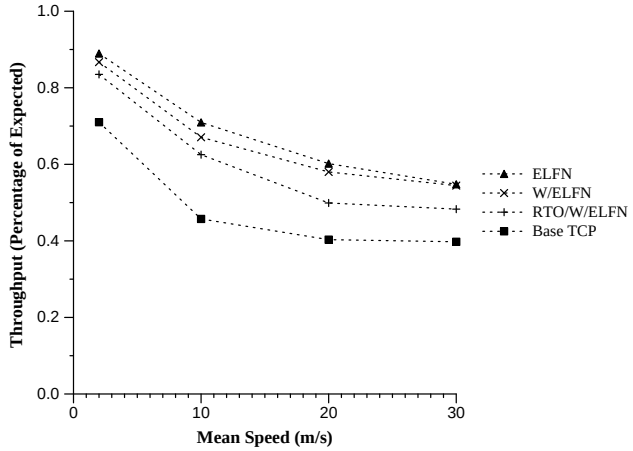


Figure 13: Performance comparison of different window and RTO modifications in response to ELFN.

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 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 (6s 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 network simulated 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 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 of such occurances to dramatically affect performance. However, this is supposition, and the true reason is unknown to us. We intend to explore this further in the future.

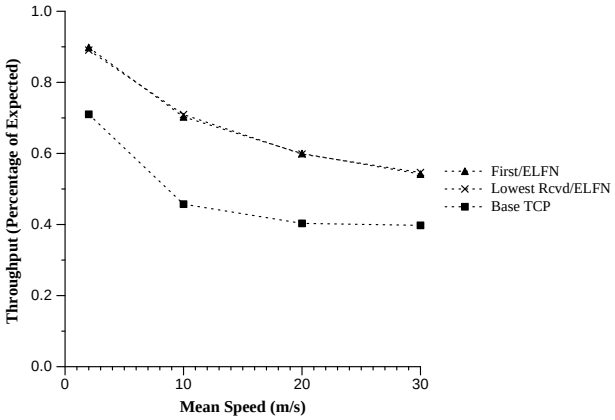


Figure 14: Performance comparison of different choices for probe packet.

Finally, we 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 received (*Lowest Rcvd/ELFN*). The first approach is intuitive, and is similar to the approach taken in SCPS-TP [8]. 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 approach 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 approaches when 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.

7 Related Work

Because routing is an important problem in mobile ad hoc networks researchers have explored several routing protocols for this environment (e.g., [9, 10, 15, 17, 18, 19, 20, 21]).

Recently, some researchers have considered the performance of TCP on multi-hop networks [14, 7]. Gerla et al. [14] investigated the impact of the MAC protocol on performance of TCP on multi-hop networks. Chandran et al. [7] proposed the TCP-Feedback (TCP-F) protocol, which uses explicit feedback in the form of route failure and re-establishment 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. [11] 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 source 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, and the other resulted 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.

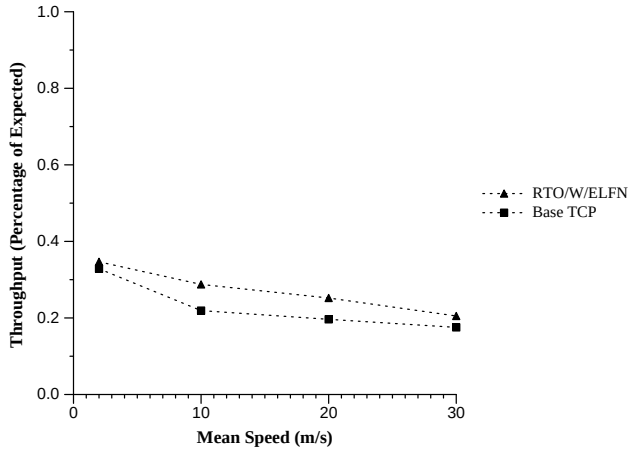


Figure 15: Performance comparison in the presence of traffic.

In the future, we intend to continue this study by looking at the performance of ELFN in congested networks. Initial results, shown in Figure 15, suggest that similar performance benefits can be expected in congested networks, such as in the uncongested network presented in this paper. Figure 15 shows a comparison of the throughput for base TCP-Reno to the use of ELFN with modifications to the RTO and congestion window, as was described in Section 6. We also intend to study the performance of other ELFN protocols, as well as the effects that other mobile ad hoc routing protocols have on TCP performance.

References

[1] H. Balakrishnan and R. Katz, "Explicit loss notification and wireless web performance," in *IEEE Globecom Internet Mini-Conference, Sydney*, Oct. 1998.

[2] 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*, Aug. 1996.

[3] 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), pp. 77–89, sep 1997.

[4] 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 Int. Conf. on Mobile Computing and Networking*, pp. 85–97, Oct. 1998.

[5] J. Broch, D. B. Johnson, and D. A. Maltz, "The dynamic source routing protocol for mobile ad hoc networks." Internet-Draft of the IETF MANET Working Group, dec 1998.

[6] R. Caceres and L. Iftode, "Improving the performance of reliable transport protocols in mobile computing environments," *IEEE Journal on Selected Areas in Communications*, vol. 13, June 1995.

[7] 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.

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

[9] M. S. Corson and A. Ephremides, "A distributed routing algorithm for mobile wireless networks," *ACM J. Wireless Networks*, vol. 1, no. 1, pp. 61–81, 1995.

[10] B. Das, E. Sivakumar, and V. Bhargavan, "Routing in ad-hoc networks using a virtual backbone." manuscript, 1997.

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

[12] K. Fall and K. Varadhan, ns *Notes and Documentation*. LBNL, August 1998. http://www-mash.cs.berkeley.edu/ns/.

[13] S. Floyd, "Tcp and explicit congestion notification," *ACM Computer Communication Review*, vol. 24, pp. 10–24, Oct. 1994.

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

[15] 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*, Aug. 1998.

[16] G. Holland and N. Vaidya, "Analysis of tcp performance over mobile ad hoc networks – part II: Simulation details and results," Tech. Rep. 99-005, Texas A&M University, Dept. of Computer Science, MS/3112, Texas A&M University, College Station, TX 77843, February 1999.

[17] 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*, Mar. 1998.

[18] 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*, Aug. 1998.

[19] S. Ramanathan and M. Steenstrup, "A survey of routing techniques for mobile communication networks," *Mobile Networks and Applications*, pp. 89–104, 1996.

[20] 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*, Oct. 1998.

[21] C. Toh, "A novel distributed routing protocol to support ad-hoc mobile computing," *Wireless Personal Communication*, Jan. 1997.