# Quantitative Lessons From a Full-Scale Multi-Hop Wireless Ad Hoc Network Testbed

David A. Maltz Josh Broch David B. Johnson
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

http://www.monarch.cs.cmu.edu/

*Abstract*—**This paper presents preliminary quantitative results from data collected during runs of our multi-hop wireless ad hoc network testbed. The network successfully carried a composite workload including voice, bulk data, and real-time data. Careful analysis of recorded runs highlights radio propagation issues that network protocols will need to address in the future.**

## I. Introduction

During the 7 months from August 1998 to February 1999, we designed and implemented a full-scale physical testbed [13] to enable the evaluation of ad hoc network performance in the field. From February through April, the testbed was used to demonstrate the potential of ad hoc networking to our sponsors and as a research tool to experiment with the carrying capacity and behavior of a fully-deployed network.

Each node in the testbed uses the Dynamic Source Routing (DSR) protocol [3], [12] to find and maintain routes to the other nodes in the testbed, and the entire testbed is integrated into the existing Internet infrastructure. We also implemented an extensive set of monitoring tools, which allow the motion and detailed protocol activity on each of the nodes to be analyzed, and a series of traffic generators to stress the network.

This paper briefly describes the results of our initial experiments on the testbed, and the considerable effect that real-world radio propagation had on the protocols in the network. The paper is not the final performance analysis of the testbed or of the DSR protocol, although such work is in progress. The quantitative numbers reported in later sections of this paper are intended to serve three purposes. First, they validate the architectural decisions made in constructing the protocol implementation. Second, they provide protocol designers with more data points on the characteristics of the outdoor wireless environment in which actual protocols for many uses must run. Finally, they point out several interesting consequences of real-world radio propagation.

This work was supported in part by the National Science Foundation (NSF) under CAREER Award NCR-9502725, by Caterpillar Corporation, and by the Air Force Materiel Command (AFMC) under DARPA contract number F19628-96-C-0061. David Maltz was also supported under an Intel Graduate Fellowship. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of NSF, AFMC, DARPA, Caterpillar, Intel, Carnegie Mellon University, or the U.S. Government.

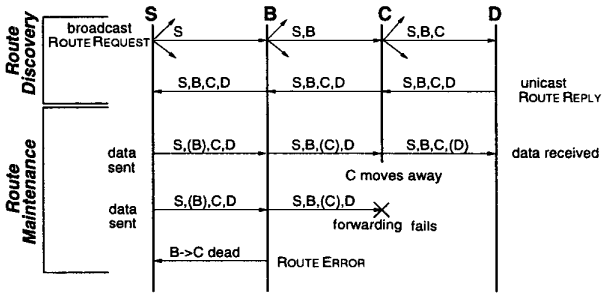


Fig. 1. Basic operation of the DSR protocol showing the building of a source route during the propagation of a ROUTE REQUEST, the source route's return in a ROUTE REPLY, its use in forwarding data, and the sending of a ROUTE ERROR upon forwarding failure. The next hop is indicated by the address in parentheses.

## II. DSR Overview

The Dynamic Source Routing protocol (DSR) [3], [7], [8] is based on source routing, such that the originator of each packet determines an ordered list of nodes through which the packet must pass while traveling to the destination. The key advantage of a source routing design is that intermediate nodes do not need to maintain up-to-date routing information in order to route the packets that they forward, since the packet's source has already made all of the routing decisions. This fact, coupled with the entirely *on-demand* nature of the protocol, eliminates the need for the periodic route advertisement and neighbor detection packets present in other protocols. Although DSR uses source routes, most packets do not need to incur the overhead of carrying an explicit source route header [3], [11].

The DSR protocol consists of two mechanisms: Route Discovery and Route Maintenance. Route Discovery is the mechanism by which a node **S** wishing to send a packet to a destination **D** obtains a source route to **D**. To reduce the cost of Route Discovery, each node maintains a Route Cache of source routes it has learned or overheard. Route Maintenance is the mechanism by which a packet's originator **S** detects if the network topology has changed such that it can no longer use its route to the destination **D** because some of the nodes listed on the route have moved out of range of each other. Figure 1 shows the basic operation of the DSR protocol.

To perform a Route Discovery, the source node **S** locally broadcasts a ROUTE REQUEST packet with the Time-to-Live field of the IP header initialized to 1. This type of ROUTE

0-7803-6596-8/00/$10.00 © 2000 IEEE

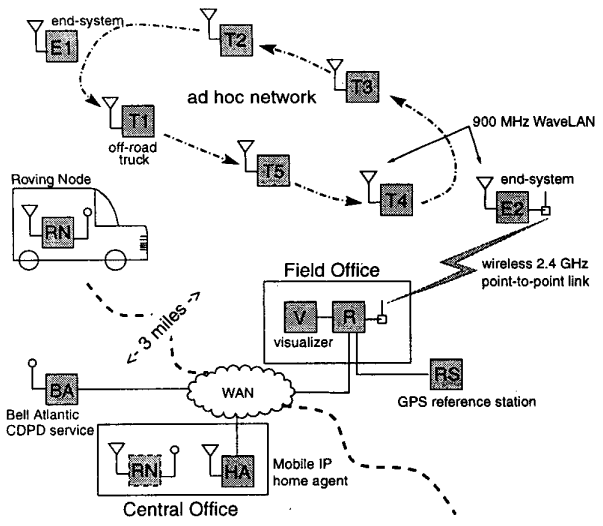


Fig. 2. Logical overview of the testbed network.

REQUEST is called a non-propagating ROUTE REQUEST. It allows node **S** to inexpensively query the Route Caches of each of its neighbors for a route to the destination, and it optimizes the case in which the destination is directly reachable. If no REPLY is returned within the nominal one-hop round trip time, node **S** transmits a propagating ROUTE REQUEST that is flooded through the network in a controlled manner and is answered by a ROUTE REPLY packet from either the destination node or another node that knows a route to the destination.

Route Maintenance is performed only when a node is attempting to forward a packet. If the packet cannot be successfully forwarded to the next-hop indicated in the packet's source route, Route Maintenance declares that link in the source route to be broken, and notifies the packet's originator **S** with a ROUTE ERROR packet. The originator **S** can then attempt to use any other route to **D** that is already in its Route Cache, or can invoke Route Discovery again to find a new route.

## III. TESTBED OVERVIEW

The design goal of the testbed was to challenge the network protocols to the point where they were stressed, by subjecting them to higher rates of topology change than previous testbeds had explored [2], [10]. With the vehicles, radios, and site used in our testbed, we forced the protocols to operate in an environment in which *all* links between nodes change status at least every 220 seconds. Ignoring the additional factor of packet loss due to wireless errors, on average, the network topology changed every 4 seconds.

Figure 2 shows a logical view of the ad hoc network testbed. The actual ad hoc network is comprised of 5 moving car-mounted nodes, labeled **T1**-**T5**, and 2 stationary nodes, labeled **E1** and **E2**. Each of these nodes communicates using 900 MHz WaveLAN-I radios. These radios do not implement the IEEE 802.11 MAC protocol [6], since at the time the testbed was built, the WaveLAN-IEEE radios were not available. The ad hoc network is connected to a *field office* using a 2.4 GHz point-to-point wireless link over a distance of about 700 m. This point-to-point link does not interfere with the 900 MHz radio interfaces on the individual ad hoc network nodes.

At the field office is a router **R** that connects both the ad hoc network and an IP subnet at the field office back to the *central office* via a wide-area network. The visualizer node **V** is used to monitor the status of the ad hoc network, and the GPS reference station (**RS**), located on the roof of the field office, is responsible for sending differential real-time kinematic (RTK) GPS corrections to nodes in the ad hoc network.

The central office is home to a *roving node* (**RN**) that drives between the central office and the ad hoc network. Node **HA** provides Mobile IP home agent services [14] for the roving node so that it is able to leave the central office and still maintain connectivity with all of the other nodes in the testbed.

During a typical experiment, which we call a *run*, the drivers of each of the cars follow a set course at speeds varying from 25 to 40 Km/hr (15 to 25 miles per hour). Each run lasts for between 30 and 120 minutes. The road we use is open to general vehicle traffic and has several Stop signs, so the speed of each node varies in a complex fashion, just as it would in any real network. Likewise, the nodes are constrained to move along the paved surfaces of the site. This prevents us from testing the arbitrary topologies used in some theoretical simulations on abstract flat planes, but enables us to evaluate the performance we can expect in a real application.

During each run, the network was subjected to the composite workload shown in Table I, consisting of synthetic voice calls, bulk data transfer, location dependent transfers, and real-time data. The workload includes: each node making one voice call to every other node once per hour; each node transferring a data file to every other node once per hour; each moving node (**T1**-**T5**) making a location-dependent transfer to **E1** when located within 150 m of **E1**; multicast differential RTK GPS corrections; and real-time situational awareness data sent by our Position and Communication Tracking daemon (PCTd).

## IV. LAYER 3 MECHANISMS FOR ACKNOWLEDGMENTS AND RETRANSMISSION

Since the WaveLAN-I radios do not provide link-layer reliability, we implemented a hop-by-hop retransmission and acknowledgment scheme within the DSR layer that provides the feedback necessary to drive DSR's Route Maintenance mechanism. One interesting aspect of our ARQ scheme was the use of passive acknowledgements [9], which significantly reduces the number of acknowledgement packets transmitted when compared to acknowledgment schemes that acknowledge every packet (e.g., IEEE 802.11 [6]).

### A. Implementation Overview

Our implementation utilizes passive acknowledgments whenever possible, meaning that if a packet's sender hears the next hop forward the packet, it accepts this as evidence that the packet was successfully received by that next hop. If a node **A** fails to receive a passive acknowledgment for a particular packet that it has transmitted to some next hop **B**, then **A** retransmits the packet, but sets a bit in the packet's header to request an explicit acknowledgment. Node **A** also requests an explicit acknowledgment from **B** if **B** is the packet's final destination, since in

TABLE I
LOAD OFFERED TO THE NETWORK BY NODES IN THE TESTBED.

| Application | Rate | Protocol | Size |
|---|---|---|---|
| *Voice* | 6/hour/node | UDP | Average of 180 kbytes |
| *Data* | 5/hour/node | TCP | 30, 60, or 90 kbytes |
| *Location-dependent* | When near **E1** | TCP | Average of 150 kbytes |
| *GPS* | 1 pkt/sec multicast | UDP | 150 bytes |
| *PCTd* | 1 pkt/sec/node unicast | UDP | 228 bytes |

this case, **A** will not have the opportunity to receive a passive acknowledgment from **B**. To avoid the inefficiencies of a stop-and-wait ARQ scheme, node **A** uses a buffer to hold packets it has transmitted that are pending acknowledgement and an identifier based on the IP ID field [15] to match acknowledgements with buffered packets.

This acknowledgement procedure allows **A** to receive acknowledgments from **B** even in the case in which the wireless link from **A** to **B** is unidirectional, since explicit acknowledgements can take an indirect route from **B** to **A**. During an average run, 90 percent of the acknowledgements used a direct one-hop route, and 10 percent of the acknowledgements were sent over routes with multiple hops. While this strongly suggests the presence of unidirectional links in the network, it does not support a conclusion that 10 percent of the packets travel over a unidirectional link. Once a multiple-hop route for acknowledgements is discovered, it may continue to be used for some period of time even after the direct route begins working again.

When performing retransmissions at the DSR layer, we also found it necessary to perform duplicate detection so that when an acknowledgment is lost, a retransmitted packet is not needlessly forwarded through the network multiple times. The duplicate detection algorithm used in our implementation specified that a node should drop a received packet if an identical copy of the packet was found in a buffer awaiting either transmission or retransmission. We found that this simple form of duplicate prevention was sufficient, and that maintaining a separate history of recently seen packets was not necessary.

### B. Packet Loss Rate

When our testbed network operated without the layer 3 acknowledgment and retransmission scheme, the average packet loss rate over a single hop was measured as 11 percent. With the ARQ scheme described, the average loss rate over a single hop dropped to 5 percent. The losses are highly correlated with position, however, and demonstrate the highly variable nature of wireless propagation due to scattering, multi-path, and shadowing effects in the real world. While future research could experiment with attempting to reduce the loss rate by requiring a greater signal strength from the packets that are received before accepting the existence of a link, the frequent occurrence of Rayleigh fades on the order of 10 dB or more argues that significant numbers of packets may still be lost.

### C. Heuristics for Selecting Timeout Values

Early in the design of our retransmission mechanism, we found that contention for the wireless medium produced enough variance in the Round Trip Time (RTT) between neighboring nodes that using a fixed value for the retransmission timer was not practical and that *adaptive* retransmission timers were required.

Our initial implementation of an adaptive retransmission timer employed the scheme used by TCP where a smoothed RTT estimator (`srtt`) and a smoothed mean deviation (`rttvar`) are maintained independently for each next hop to which a node is communicating. The retransmission timeout (RTO) is then computed as:

$$RTO = srtt + 4 * rttvar$$

Unfortunately, the variance in RTT prevented this implementation from performing adequately. Frequently, the RTO would not adapt quickly enough to congestion in the network and packets would be retransmitted unnecessarily, creating even more congestion. It also suffered from the fact that the RTO to each next hop was computed independently, while the need to defer transmissions due to congestion is common across all neighbors accessed via the same network interface.

We found that several simple methods of reacting to increasing congestion did not work. For example, if retransmission timeouts are treated as a RTT sample of twice the current RTT, the value of the retransmission timer tends to diverge and remain pegged at its maximum value, even after congestion has subsided.

We developed a successful retransmission timer algorithm by including a heuristic estimate of the level of local congestion, so that the retransmission timer could react quickly to changes. One of the simplest ways for a node to measure congestion in the network is to look at the length of *its own* network interface transmit queue. Specifically, if more than 5 packets are found in the interface transmit queue — meaning that congestion is starting to occur — we increase the value of the retransmission timer 20 ms for each packet in the queue. Assume that there are $N$ packets in the network interface queue. For $N \leq 5$, the retransmission timeout is computed as before:

$$RTO = srtt + 4 * rttvar$$

However, for $N > 5$, the retransmission timeout is computed as:

$$RTO = srtt + 4 * rttvar + ((N - 5) * 20 \text{ ms})$$

This heuristic allows the retransmission timer to increase quickly during periods of congestion and then return just as quickly to its computed value once the congestion dissipates. In 4710 measurements over several runs, approximately 75% of the packets transmitted use the minimum retransmission timer value of 50 ms. However, for the other 25% of the packets, the retransmission timer adjusted itself to values between 60 ms and 920 ms. The wide range indicates that an adaptive retransmission scheme is required for good performance if acknowledgments are implemented above the link layer.

## V. Jitter in Inter-packet Spacing

Isochronous communications, such as interactive voice and remote telemetry, are extremely sensitive to packet jitter. In order to evaluate how well isochronous communications would work across our testbed network, we evaluated the jitter experienced by the synthetic audio traffic sent as part of the composite workload described in Table I. Each audio connection consisted of alternating simplex packet streams between the communicating parties, and each packet stream consisted of 250-byte UDP packets sent 8 times per second, giving an average bit rate of 16 Kb/s. This traffic pattern was chosen to model the Push-To-Talk mobile radios commonly used on construction sites.

During an average run, 98,000 voice packets are originated, which represents a total of 3.4 hours of voice. Of these 98,000 packets, 3.8% are lost in the network. The testbed does not contain any special handling rules for the voice packets, so the packets are retransmitted according to the same mechanism described in Section IV. The jitter, defined as the variation in inter-packet spacing introduced by the network, ranged from $-9.4$ s to 6.5 s. This extreme amount of jitter is rare, and typically occurs when the network is temporarily partitioned. During a partition, the voice sources continue to send data, but the packets are buffered inside the network. The result is a burst of back-to-back packet arrivals at the destination when the partition heals. As an area for future research, more sophisticated packet handling algorithms might detect these delayed but time-sensitive packets and drop them inside the network to conserve resources [11].

When the most extreme 2% of jitter samples are removed as outliers, the range of jitter drops to between $-1.04$ s and 1.02 s. The mean jitter is 0.001 s, and the standard deviation 0.143 s. Figure 3 shows the distribution of jitter samples using a histogram. The y-axis is on a log scale for clarity: each bar is 20 ms wide, and there are 10 times more packets with 0.0 s of jitter than packets with either $\pm$ 20 ms of jitter. 90% of the voice packets experience a jitter between $-0.2$ s and 0.2 s, so a playback buffer of 400 ms should be sufficient for voice communication.

## VI. The Need for Hysteresis in Route Selection

As mentioned above, the packet loss rate seen between any two nodes in the network is highly variable, depending not only on the positions of the nodes involved, but also on the movement of other objects around the nodes. In working with TCP

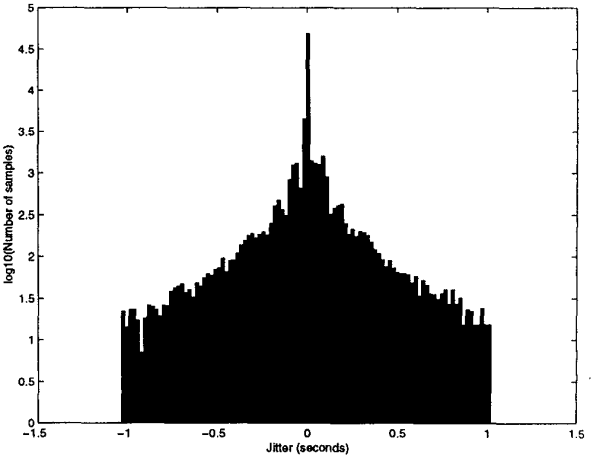


Fig. 3. Distribution of jitter. Y-axis is on a log scale for clarity.

connections carried over the testbed network, we found this variability had a disastrous effect on the bandwidth delivered by these TCP connections. Other researchers have addressed related problems in layer 4 and at the boundary between layer 3 and layer 4 [1], [5], so in this section we will concentrate on a layer 3 issue caused when radio propagation is transiently *better* than expected.

To isolate the layer 3 issue, we conducted an experiment with three nodes arranged in a linear fashion. The nodes were positioned by driving two cars in opposite directions and positioning them as far from the middle node as possible, while still allowing both of the end nodes to successfully flood ping the intermediate node with 1024-byte packets (i.e., sending ping packets as fast as possible). Once positioned, the nodes remained stationary for the remainder of the test. For the purpose of discussion, let node **A** be the TCP source, **B** the intermediate node, and **C** the TCP sink.

This is a particularly challenging scenario, not only because electromagnetic propagation is highly variable, but because the specific setup of this test introduces the hidden terminal problem. A number of times during these tests, we saw the DSR retransmission timer expire, creating ROUTE ERRORs and subsequent ROUTE REQUESTs as the nodes attempt to restore connectivity.

As described in Section II, nodes using DSR discover routes to other nodes by first sending a non-propagating ROUTE REQUEST that the target node will answer with a ROUTE REPLY if can directly receive the originator's REQUEST. If the originator does not receive a ROUTE REPLY within 30 ms, it sends a propagating ROUTE REQUEST that floods through the network in a controlled fashion to discover multi-hop routes to the target.

In order to obtain baseline performance metrics, we used our `macfilter` tool [13], which allows us to create a synthetic propagation environment among nodes in a laboratory setting. We first made 5 independent transfers of 1 MB each from node **A** to node **C**. Over these 5 transfers, TCP averaged 0.50 Mb/s (61 KB/s) with a standard deviation of 0.079 Mb/s. When the same transfers were performed in the field, however, the average data rate was 0.12 Mb/s (14.65 KB/s) with a standard deviation

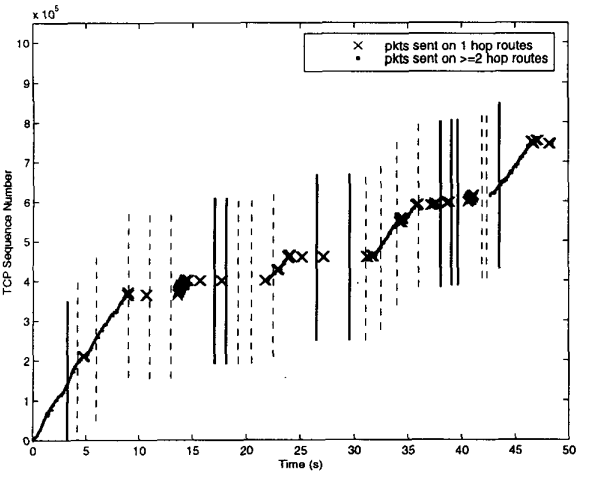


Fig. 4. A TCP sequence number plot for a 1 MB transfer over a two-hop route. The vertical lines indicate the times at which the TCP source initiated Route Discovery; the dashed lines indicate the times at which only a non-propagating ROUTE REQUEST was transmitted and the solid lines indicate both a non-propagating and a propagating ROUTE REQUEST.

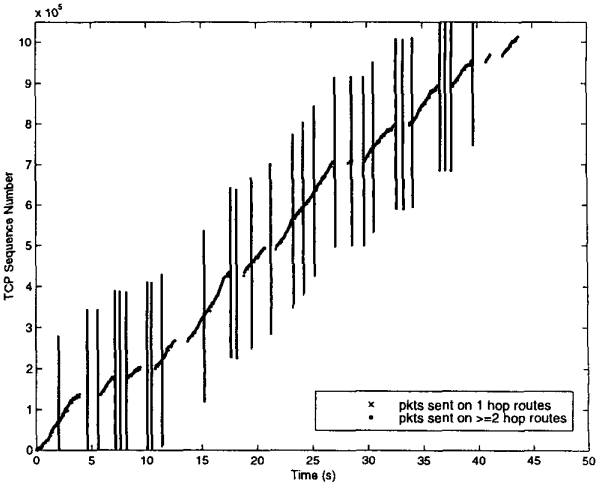


Fig. 5. A TCP sequence number plot for a 1 MB transfer over a two-hop route when the *macfilter* utility was used on both the source and destination nodes to prevent the use of single-hop routes. The vertical lines indicate the times at which the TCP source initiated Route Discovery.

of 0.025 Mb/s — only 25% of the throughput measured in the lab. In fact, some of the 1 MB data transfers, which were set up to last for a maximum of 50 seconds, timed out before the entire megabyte could be transfered. In these cases, we report the average data rate for the 50-second duration of the connections.

The time-sequence number plot from a typical two-hop connection in the testbed is depicted in Figure 4. Sequence numbers marked with a small dot were transmitted using the two-hop route **A**→**B**→**C**, while sequence numbers marked '×' were transmitted using the one-hop route **A**→**C**. The dashed vertical lines in the figure indicate when the TCP source **A** performed a Route Discovery consisting only of a non-propagating ROUTE REQUEST, and the solid vertical lines indicate when a Route Discovery consisting of both a non-propagating and a propagating ROUTE REQUEST occurred (by the rules of Route Discovery, if the non-propagating REQUEST returns a REPLY the propagating REQUEST is not sent).

The TCP connection in Figure 4 made very good progress for the first 9 seconds of the connection, using almost exclusively a two-hop route. However, during the time interval from 9 s to 22 s, the connection makes almost no progress, sending about 30 KB in this 13 s interval. After processing a ROUTE ERROR at t = 9 s, the TCP source (node **A**) initiates a ROUTE DISCOVERY. The non-propagating ROUTE REQUEST is answered directly by node **C**, causing **A** not to send a subsequent propagating ROUTE REQUEST and thus to use a single-hop route to node **C**. The poor quality of this single-hop link leads to repeated errors and Route Discovery attempts. Finally, at t = 18 s, node **A**'s non-propagating ROUTE REQUEST fails to return any REPLYs and so **A** transmits a propagating REQUEST. This results in the discovery of both the single-hop route and the two-hop route through intermediate node **B**. By this time, TCP has backed off and does not offer the next packet to the network until t = 22 s. Node **A** attempts to use the one-hop route that it discovered, finds that it does not work well, removes the one-hop route from its Route Cache, and begins using the two-hop route. At this time, the connection again starts making progress. The same scenario of repeated attempts to use a one-hop route occurs again from t = 25 s to 32 s and from t = 35 s to 43 s.

This scenario illustrates an important challenge for ad hoc network routing protocols and argues strongly that all routing protocols need some ability to remember which recently used routes have been tried and found not to work. Even traditional distance vector style protocols are subject to this problem as they attempt to minimize a single metric — usually hop count.

Considering the three-node scenario discussed above, if **A**, **B**, and **C** were all participating in a distance vector routing protocol, **A** would sometimes hear advertisements from node **C**. Since the direct route to **C** is more optimal in terms of hop count than the route through the intermediate node **B**, **A** would attempt to send all of its packets directly to **C** until that direct route timed out. In other words, without some type of local feedback or other hysteresis, **A** will often try to send its packets directly to **C**, effectively black-holing most of these packets since that link is so unreliable. Protocols such as Signal Stability Based Routing (SSA) [4] may behave much better in this scenario.

To evaluate the potential gain of having a mechanism that would prevent the repeated use of the poor direct route from **A** to **C**, we emulated perfect routing information by using our `macfilter` to eliminate the discovery of 1-hop routes. Figure 5 shows the time-sequence plot for a 1 MB transfer in the field using this "perfect routing." The flat plateaus are no longer present, and the throughput is 30% higher. The remaining Route Discoveries are triggered when packets are repeatedly lost due to variation in wireless propagation, and techniques such as notifications to the TCP module could be used to prevent the TCP stalls that follow these Route Discoveries.

We are presently considering three ways to implement such a mechanism in DSR. One solution would be for DSR to cache information about each link for which it receives a ROUTE ERROR. This negative information could be timed out after a

period based on the estimated rate of link fluctuation, but would prevent DSR from repeatedly attempting to use a poor quality link. The drawback of this solution is the difficulty of designing a strategy to pick a reasonable timeout value.

While not a generic technique, in networks where each node knows its position (e.g., due to the use of GPS as in our network) communicating nodes could use the location information propagated by the other nodes to model the position of their correspondents. If the link **A** to **C** is found to be bad, DSR could retain that negative information in its cache until it finds that either node **A** or **C** has changed position in some reasonably significant way.

The third and more sophisticated approach would combine the signal strength at which the node received ROUTE REPLYs, the position of the nodes, and the mobility pattern of the nodes to estimate the probability of successful communication over a particular route [16].

## VII. CONCLUSIONS

We have created a testbed for ad hoc network research, featuring 2 stationary nodes, 5 car-mounted nodes that drive around the testbed site, and 1 car-mounted roving node that enters and leaves the site. Packets are routed between the nodes using the DSR protocol, which also seamlessly integrates the ad hoc network into the Internet via a gateway. The use of Mobile IP permits nodes to roam transparently between the ad hoc network and normal IP subnets.

In preliminary analysis of data from runs of the testbed, we have measured the jitter introduced by the network and found that even under a full load and without any special QoS handling, the network can support a Push-To-Talk voice system. We have also explained how our novel heuristic for the DSR-layer packet acknowledgement and retransmission scheme allows nodes to rapidly adapt to changing levels of network congestion. Finally we have demonstrated the need for hysteresis in the selection of routes that are used in ad hoc networks to prevent the use of routes that exist only transiently.

## VIII. ACKNOWLEDGEMENTS

The Monarch Project ad hoc network testbed was the product of the work of many people, but special recognition is due to Jorjeta Jetcheva, Qifa Ke, and Ben Bennington.

We are also grateful for the efforts of the other members of the research team, including Ratish Punnoose, Pavel Nikitin, Dan Stancil, Satish Shetty, Michael Lohmiller, Yih-Chun Hu, Sam Weiler, and Jon Schlegel.

## REFERENCES

[1] Bikram S. Bakshi, P. Krishna, N. H. Vaidya, and D. K. Pradham. Improving the Performance of TCP over Wireless Neworks. In *Proceedings of the 17th International Conference on Distributed Computing Systems (ICDCS'97)*, pages 365–373, May 1997.

[2] David A. Beyer. Accomplishments of the DARPA Survivable Adaptive Networks SURAN Program. In *Proceedings of MILCOM'90*, 1990.

[3] Josh Broch, David B. Johnson, and David A. Maltz. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks. Internet-Draft, draft-ietf-manet-dsr-03.txt, October 1999. Work in progress.

[4] Rohit Dube, Cynthia D. Rais, Kuang-Yeh Wang, and Satish K. Tripathi. Signal Stabiljty based Adaptive Routing (SSA) for Ad Hoc Mobile Networks. *IEEE Personal Communications*, pages 36–45, February 1997.

[5] Gavin Holland and Nitin Vaidya. Analysis of TCP Performance over Mobile Ad Hoc Networks. In *The Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'99)*, pages pp. 219–230, 1999.

[6] IEEE Computer Society LAN MAN Standards Committee. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std 802.11-1999. The Institute of Electrical and Electronics Engineers, New York, New York, 1999.

[7] David B. Johnson. Routing in Ad Hoc Networks of Mobile Hosts. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, pages 158–163, December 1994.

[8] David B. Johnson and David A. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. In *Mobile Computing*, edited by Tomasz Imielinski and Hank Korth, chapter 5, pages 153–181. Kluwer Academic Publishers, 1996.

[9] John Jubin and Janet D. Tornow. The DARPA Packet Radio Network Protocols. *Proceedings of the IEEE*, 75(1):21–32, January 1987.

[10] Robert E. Kahn, Steven A. Gronemeyer, Jerry Burchfiel, and Ronald Kunzelman. Advances in Packet Radio Technology. *Proceedings of the IEEE*, 66(11):1468–1496, November 1978.

[11] David A. Maltz. Resource Management in Multi-hop Ad Hoc Networks. Technical Report CMU CS 00-150, School of Computer Science, Carnegie Mellon University, July 2000. Available from http://www.monarch.cs.cmu.edu/papers.html.

[12] David A. Maltz, Josh Broch, Jorjeta Jetcheva, and David B. Johnson. The Effects of On-Demand Behavior in Routing Protocols for Ad Hoc Networks. *IEEE Journal on Selected Areas in Communications*, 17(8):1439–1453, August 1999.

[13] David A. Maltz, Josh Broch, and David B. Johnson. Experiences Designing and Building a Multi-Hop Wireless Ad Hoc Network Testbed. Technical Report 99-116, School of Computer Science, Carnegie Mellon University, March 1999. Available from http://www.monarch.cs.cmu.edu/papers.html.

[14] Charles Perkins, editor. IP Mobility Support. RFC 2002, October 1996.

[15] J. Postel. Internet Protocol. RFC 791, September 1981.

[16] Ratish J. Punnoose, Pavel V. Nikitin, Josh Broch, and Daniel D. Stancil. Optimizing Wireless Network Protocols Using Real-Time Predictive Propagation Modeling. In *Radio and Wireless Conference (RAWCON)*, Denver, CO, August 1999.