Experiences Designing and Building a Multi-Hop Wireless Ad Hoc Network Testbed

David A. Maltz Josh Broch David B. Johnson

March 5, 1999
CMU-CS-99-116

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

In this paper, we describe our experiences building a multi-hop wireless ad hoc network of 8 nodes driving around a 700 m by 300 m site. Each node runs the Dynamic Source Routing (DSR) protocol and interfaces seamlessly with existing Internet infrastructure and the Mobile IP protocol. The issues discussed in this paper range from logistical and management issues, to protocol design and performance analysis issues. We also present an early characterization of the testbed performance, and describe a significant new challenge for ad hoc network routing protocols. The major goal of the paper, however, is to share our experiences, in the belief that they may be useful to others who attempt to build other ad hoc network testbeds.

This work was supported in part by the National Science Foundation (NSF) under CAREER Award NCR-9502725, by the Air Force Materiel Command (AFMC) under DARPA contract number F19628-96-C-0061, and by the AT&T Foundation under a Special Purpose Grant in Science and Engineering. 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, the AT&T Foundation, Intel, Carnegie Mellon University, or the U.S. Government.

Keywords: multi-hop ad hoc networks, Dynamic Source Routing (DSR), mobile wireless network testbed

Contents

1 Introduction

During the 7 months from August 1998 to February 1999, we designed and implemented a full-scale physical testbed to enable the evaluation of ad hoc network performance in the field. The last week of February and the first week of March included demonstrations of this testbed to a number of our sponsors and partners, including Lucent Technologies, Bell Atlantic, and DARPA. Though the process was very exciting for us, we encountered numerous difficulties ranging from the technical to the mundane. This paper describes our approach, the obstacles we encountered, and our current solutions. Many of the problems that arose and the issues that are described in this paper appear painfully obvious *after* they are identified. However, as the interest in multi-hop ad hoc networks continues to grow, more and more groups will attempt to build and deploy similar testbeds and networks, and we hope our experiences will enable them to avoid some of the difficulties we faced.

Over the past several years, much research effort has been focused on the design and specification of routing protocols for multi-hop wireless ad hoc networks. There are now at least ten proposals for such protocols currently under consideration by the Mobile Ad Hoc Networks (MANET) Working Group of the Internet Engineering Task Force (IETF). Before large ad hoc networks can be deployed, testbeds must be created to profile the performance of these protocols in the real world.

This paper is written as a design study with the goal of identifying a series of issues that others should consider when designing their own multi-hop ad hoc network testbeds. In particular:

- We identify several tools we wrote to assist in the debugging, validation, and analysis of the testbed. We argue that one tool, called `macfilter`, was critical to our success, and should be implemented in other testbeds as well. We explain the rationale and cost/benefits of the other tools, including changes we had to make to standard tools before they would work as desired in the testbed environment.
- We present an overview of how we architected the Dynamic Source Routing (DSR) protocol into the existing BSD Unix network stack. It is our hope that this will prove useful to other protocol architects who wish to implement on-demand routing protocols in traditional network stacks. We also explain the implementation and validation of a network-layer hop-by-hop acknowledgement scheme, including a novel heuristic for setting the retransmission timer.
- We describe the methodology we used to perform an initial characterization of the propagation environment at the testbed site, as well as initial testing of the protocols. This paper is *not* a performance analysis of the testbed or of the DSR protocol (though such work is in progress). The quantitative numbers reported in later sections of this paper show that the testbed is working and validate that the DSR implementation is behaving reasonably. They also point out several interesting consequences of the outdoor wireless environment.

Even if the designers of future testbeds choose to make different decisions than we did, the list of issues contained in this paper should be valuable for identifying the tasks requiring attention so that staff assignments can be adequately scheduled. To place the testbed into context, the next section describes the goals that motivated our testbed.

2 Goals of the Testbed

In designing the testbed, our primary goal was to build a platform that would enable basic research on the behavior of a real implementation of ad hoc network protocols operating with truly mobile nodes in a outdoor environment. We wanted the testbed to operate in an outdoor setting, since many currently envisioned applications of ad hoc networks operate outdoors [17], and this environment is inherently more unpredictable than an in-building environment. Changes in weather, the motion of cars and pedestrians, and the presence of buildings and hills all effect the propagation of radio signals. These factors constitute challenges that a deployed ad hoc network will face, and so we wanted to experience them in our testbed. Development of an indoor ad hoc network testbed is a subject for further study.

Another goal of the testbed was to push the protocols to the point where they nearly broke, by subjecting them to higher rates of topology change than previous testbeds had explored [16, 1]. With the vehicles, radios, and site used in our testbed, we forced the protocols to adapt to an environment in which *all* links between nodes changed 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.

There have been several prior efforts to build ad hoc network testbeds or large experiments, including work by the SURAN Project [1], the WINGS Project [10], Task Force XXI [24], and BBN. Very little has been published about the specifics of these testbeds, though based on available literature, they are very different from ours, both in terms of radio technology and network design.

3 DSR Protocol Overview

The ad hoc network routing protocol used in our testbed is the Dynamic Source Routing protocol (DSR) [12, 13, 2]. Figure 1 shows the basic operation of the DSR protocol, which 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 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 REQUEST is called a non-propagating ROUTE REQUEST and allows node **S** to inexpensively query the route caches of each of its neighbors for a route to the destination. If no

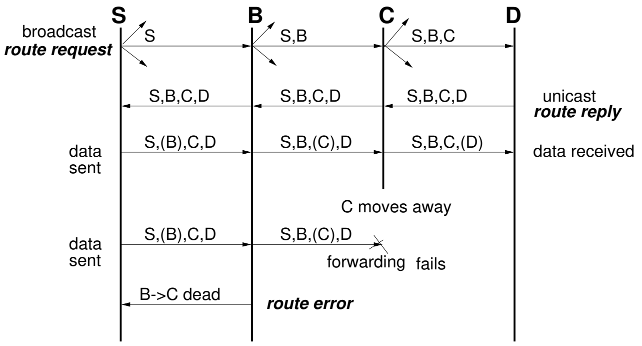


Figure 1 Basic operation of the DSR protocol showing the building of a source route during the propagation of a ROUTE REQUEST, the source route's return in a ROUTE REPLY, its use in forwarding data, and the sending of a ROUTE ERROR upon forwarding failure. The next hop is indicated by the address in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

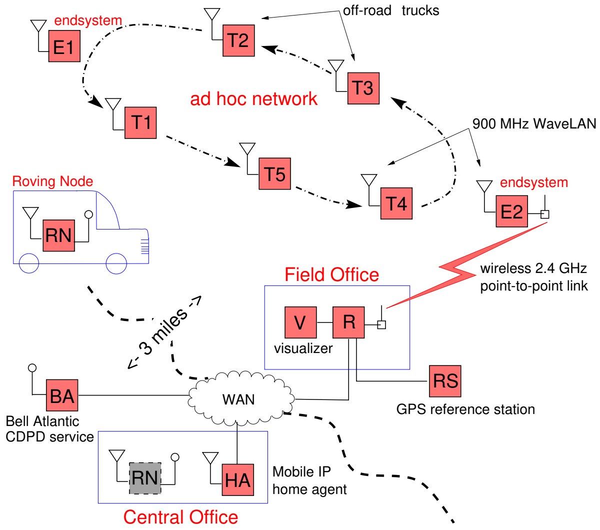


Figure 2 Logical overview of the testbed network.

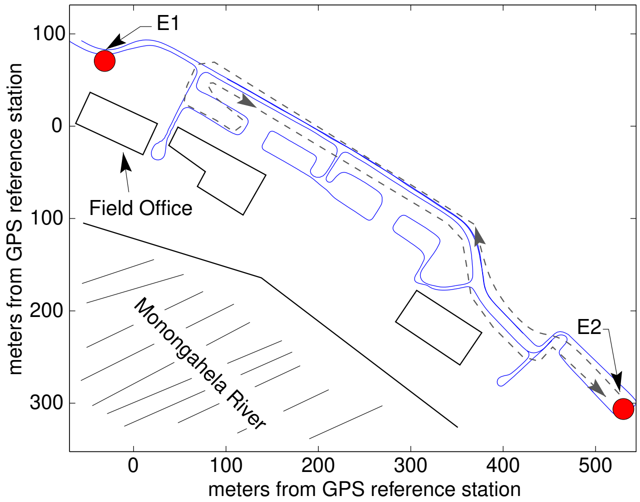


Figure 3 Map of the test bed site showing the endpoint locations and the typical course driven by the nodes.

REPLY is returned within the nominal one-hop round trip time[1], 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. To reduce the cost of Route Discovery, each node maintains a cache of source routes it has learned or overheard, which it aggressively uses to limit the frequency and propagation of ROUTE REQUESTs.

Route Maintenance is the mechanism by which a packet's sender **S** detects if the network topology has changed such that it can no longer use a known route to the destination **D** because two nodes listed in the route have moved out of range of each other. When Route Maintenance indicates a source route is broken, **S** is notified with a ROUTE ERROR packet. The sender **S** can then attempt to use any other route to **D** already in its cache or can invoke Route Discovery again to find a new route. Since the Lucent Technologies WaveLAN-I radios [25] used in our testbed do not provide any link-layer acknowledgment that a transmitted packet was successfully received, Route Maintenance in our implementation is driven by the DSR acknowledgment and retransmission mechanism described in Section 6.2.

4 Testbed Overview

Figure 2 shows a logical view of the ad hoc network testbed. The actual ad hoc network is comprised of 5 moving nodes, labeled **T1**-**T5**, and 2 stationary nodes, labeled **E1** and **E2**, that communicate using 900 MHz WaveLAN radios. 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 radio interfaces on the individual ad hoc network nodes.

[1]In our testbed, we used 30 ms as the timeout for non-propagating ROUTE REQUESTs.

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 (Section 6.5.2), and the GPS reference station (**RS**) located on the roof of the field office is responsible for sending differential 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 [22] 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 moving nodes follow the course shown in Figure 3 at speeds varying from 25 to 40 Km/hr (15 to 25 miles per hour). The road we used is open to general vehicle traffic and has several Stop signs, so the speed of each node varies over time, just as it would in any real, deployed network.

We chose this configuration for our testbed as it is similar to a variety of possible military and commercial scenarios. For example, in a military scenario, the central office could represent a Tactical Operations Center, the field office a battlefield command post, **E2** could be a munitions dump, and the moving vehicles **T1**-**T5** a convoy of trucks carrying ammunition to a firing position at **E1**. In a civil disaster relief scenario, the offices could again represent regional and local command centers, and the trucks **T1**-**T5** could be ambulances or water tankers moving between the disaster site **E1** and the staging/treatment area **E2**. In all cases, the roving node **RN** can represent a supervisor sent to inspect the scene, a service truck sent to repair one of the nodes, or even an uninvolved node making use of the ad hoc network for improved connectivity as it wanders through the area.

4.1 Design Decisions

In choosing how many nodes to use in the testbed, we strove to include enough nodes to create interesting multi-hop behavior, while keeping the equipment and personnel requirements for operating the nodes small enough to be feasible.

The geographical size of the testbed (700 m by 300 m), in turn, was set as a balance between the number of nodes available (7, excluding the roving node **RN**) and the range of the radios used by the nodes (approximately 250 m). Increasing the range of the radios would allow us to run the testbed over more varied terrain, but would require nodes to move faster to maintain the same rate of topology change. Even with the radios that we chose, however, we found that we had to mount the nodes inside cars in order to move them in and out of range of each other in reasonable lengths of time.

We derived several benefits from our solution to the constraints outlined above. Working outdoors allowed us to use GPS (Global Positioning System) to accurately track the position of the nodes, which was critically helpful in characterizing the network's performance. Another advantage of keeping the range of the radios small and working outdoors is the possibility of climbing to a high point from which the entire network can be seen at one time. The importance of visualizing the entire network can not be overstated, especially during the early stages of development when the network was not working properly.

4.2 Network Configuration

All communication among the ad hoc network nodes, **T1**-**T5**, **E1**, and **E2**, is routed by the Dynamic Source Routing (DSR) protocol [12, 13, 2]. Although the DSR protocol operates at the IP layer of the network stack (OSI layer 3) and permits interoperation between different physical interfaces, DSR conceptually operates as a virtual link layer just under the normal IP layer.

Nodes **T1**-**T5**, **E1**, and **E2** are assigned IP addresses from a single subnet, with **E2** acting as a gateway between the Internet and the ad hoc subnet. **E2** was manually configured to use the DSR protocol for communication on one network interface (the 900 MHz WaveLAN link), and to use normal IP routing over the other interface (the 2.4 GHz point-to-point link to its default router **R**). Packets from nodes in the Internet destined to addresses in the ad hoc subnet are routed by normal means to **E2**, which has a statically configured route directing them out the network interface to the ad hoc network. Once forwarded into the ad hoc network by **E2**, DSR takes care of routing the packets to their final destination, which often requires multiple hops inside the ad hoc network. As explained in Section 6.1, nodes in the ad hoc subnet (i.e., **T1**-**T5** and **E1**) did not have to be configured to use **E2** as a default router: when nodes in a DSR ad hoc network send packets to nodes not in the ad hoc network, the DSR protocol itself automatically routes the packets to the nearest gateway (**E2** in this case), where they are forwarded into the Internet. The gateway node, **E2**, also provides Mobile IP foreign agent services to any Mobile IP nodes that visit the ad hoc network.

The roving node **RN** has available several methods for connecting to the Internet, and uses Mobile IP [22] to choose the best method as it drives around the city. **RN** is normally within range of the WaveLAN network at the central office, and its WaveLAN network interface carries an IP address belonging to the central office subnet. When **RN** is roving away from the central office, it uses Mobile IP to register a *care-of address* with its *home agent* on the central office subnet. While **RN** has a care-of address registered with the home agent, the home agent intercepts packets destined to **RN**, and tunnels each to the care-of address using encapsulation.

When **RN** cannot use its primary WaveLAN interface because it is not in range of any other WaveLAN radios, it uses its CDPD modem to connect to Bell Atlantic Mobile's CDPD service, and registers its CDPD IP address with its home agent. Once **RN** realizes it is in range of a DSR network, it can use the DSR protocol to communicate directly with the other nodes in the ad hoc network. To enable packets from nodes outside the DSR network to reach **RN**, it registers itself with its home agent via the foreign agent at **E2**, just as in normal Mobile IP. When **E2** receives a tunneled packet, it checks to see if the packet is destined to a node registered as visiting the ad hoc network. If so, **E2** routes the packet to the visiting node using DSR.

4.3 Node Equipment

Nodes **T1**-**T5** were each implemented by a rented car carrying an IBM Thinkpad 560X notebook. The Thinkpads were each mounted in a home-built rack carried in the front passenger's seat of the car, which also housed a GPS unit and 12VDC to 110VAC power converter used to power the equipment. Although tightening the 16 bolts that hold together each rack was blister producing, experience has shown the racks are a worthwhile investment as they prevent the equipment in the cars from becoming jumbled and eliminate the risk of cable connections coming undone.

The 5 moving nodes (**T1**-**T5**) each carry a 900 MHz Lucent WaveLAN-I [25] radio connected to a 6 dB omni-directional antenna. These antennas are mounted on a rack on the roof of the cars, about 9 feet above the ground. The WaveLAN-I radio is a Direct Sequence Spread Spectrum device with a raw capacity of 2 Mb/s, and a 250 m nominal transmission range. The WaveLAN-I uses a Lucent-designed CSMA/CA MAC protocol that does not include link-layer retransmissions or acknowledgments. It also does not use mechanisms like those in IEEE 802.11 [11], such as RTS-CTS, to avoid hidden terminal issues.[2] We chose the WaveLAN radio as they have a high enough bandwidth (2 Mb/s) to support both audio and data traffic. Additionally, we have a long history of working with these radios in both indoor and outdoor environments, they use unlicensed spectrum, and they are available with a jack for an external antenna.

To enable the moving nodes to determine their location, they each carry a Trimble 7400 GPS receiver with the GPS antenna mounted on the roof rack alongside the WaveLAN antenna. Each GPS receiver is capable of calculating its position to within 100 m at all times, but when provided with *correction information* from a GPS reference station can calculate its position to within 1 m or 1 cm, depending on the frequency and latency at which the correction information is provided. When receiving correction information at least once per 25 s, the GPS receiver operates in Differential mode (DGPS) and calculates positions with 1 m accuracy. When receiving correction information consistently once per second, the GPS receiver operates in Real Time Kinematic mode (RTK) and calculates positions with approximately 1 cm accuracy. In our testbed, the GPS reference station was located at the field office, and the correction information it generated was sent to the nodes as a stream of broadcast packets over the multi-hop ad hoc network.

The two end-systems shown in Figure 3, **E1** and **E2**, were located at opposite ends of the course traveled by the mobile nodes and were implemented with laptops identical to those used in the moving nodes **T1**-**T5**. Because of their location, **E1** and **E2** could be conveniently used to test connectivity across the diameter of the ad hoc network. **E1** carries a single 900 MHz WaveLAN radio with a 6 dB omni-directional antenna identical to that on the moving nodes. In contrast, **E2** serves as a router between the ad hoc network and the rest of the Internet. It communicates with nodes in the ad hoc network using a 900 MHz WaveLAN radio attached to a 6 dB yagi antenna and is linked to router **R** at the field office using a 2.4 GHz WaveLAN radio connected to a 12 dB yagi antenna. We sited **E2** at the opposite end of the course from the field office to demonstrate that the ad hoc network did not have to be close to any wired infrastructure. There is no 900 MHz WaveLAN radio in the field office, so all traffic into or out of the ad hoc network must travel through **E2**.

[2] At the time we began designing the testbed, the Lucent IEEE 802.11 product was not readily available, there were no FreeBSD device drivers supporting this hardware, and the cards worked only when used with a base-station. Thus, this product was not a viable choice for our testbed.

IP Header (next protocol = Hop-By-Hop Options)
Hop-By-Hop Options Header (next protocol = DSR Routing)
DSR Routing Header (next protocol = TCP)
TCP Header TCP data

Figure 4 Layout of headers in a typical packet in the DSR network.

The roving node, **RN**, carried a 900 MHz WaveLAN radio and could use it to join the ad hoc network by communicating with nodes **T1**-**T5**, **E1**, and **E2**. It also carried a Cellular Digital Packet Data (CDPD) modem that it could use for wide area digital packet service at 11 Kb/s provided by Bell Atlantic Mobile Systems.

Each of the nodes at the central and field offices are Intel Pentium II PCs. All nodes ran the FreeBSD 2.2.7 UNIX system, modified with our DSR [2] and Mobile IP [22] kernel extensions.

5 Protocol Implementation

Our earliest implementation of DSR began as an extension to the Address Resolution Protocol (ARP). Running at the link-layer, each Ethernet frame had a DSR source route inserted between the Ethernet header and the IP header. This solution was simple, but could not span across multiple interface types.

The desire to support heterogeneous networks pushed our implementation into the IP layer. The need to combine multiple types of DSR information together on the same packet and the desire to piggyback DSR information on data packets led us a packet format based the IPv6 extension header scheme.

5.1 Packet Format

The control messages of the DSR protocol are encoded using a series of extension headers that lie between the IPv4 header and the normal IPv4 payload (e.g., the ICMP, UDP, or TCP header and data). This enables control messages to be piggybacked on ordinary data packets when they are available, or to be sent as separate control packets. Figure 4 shows the layout of the headers in a typical packet. As in IPv6, we use three types of extension headers:

Hop-by-Hop Options: Processed by every node that receives the packet and is used to carry DSR ROUTE REPLYs, ROUTE ERRORs, and ACKNOWLEDGEMENTS. The Hop-by-Hop Options extension header is also used to carry a GPS Option that gives the physical location of packet's originator at the time when the packet was transmitted.

Destination Options: Processed only by a node whose address matches the IP Destination Address in the packet (which can be a unicast, multicast, or broadcast address). These headers are used to carry ROUTE REQUESTs.

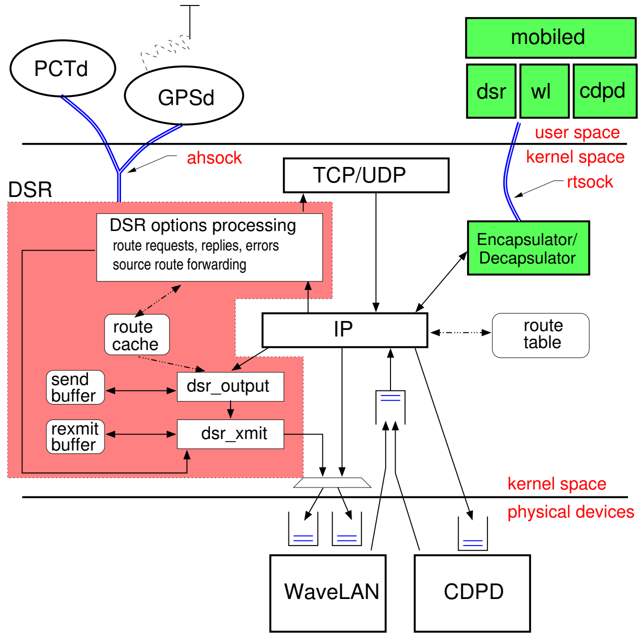


Figure 5 The internal architecture of a node showing the DSR components and the Mobile IP components.

Routing Header: Processed by a node whose address is equal to the IP Destination Address of the packet. Carries the source route describing the path that a packet must take through the network.

With the one caveat explained next, we have found the extension header scheme an elegant way to transport routing information for our on-demand routing protocol, and would recommend it to other designers. We used the extension header processing framework from the INRIA IPv6 distribution [8]. Although the code was modular and easy to work with, we found that when passing around the chain of `mbufs` that comprise a packet, we frequently had to read through the whole chain to find some header or other bit of information in the packet. Were we to implement from scratch again, we would pass along with the `mbuf` chain a structure containing all the critical information present in the packet (e.g., the next hop, whether an ACK is required, etc.), thereby enabling the code to process the packet without having to find and read the same extension header multiple times.

5.2 Outgoing Packets

Figure 5 shows the overall architecture of a node in the ad hoc network, including the DSR and Mobile IP kernel components, the user-level Mobile IP daemon, and the user-level logging utilities.

All of the code implementing the DSR protocol resides inside the kernel in a module that straddles the IP layer. Conceptually, however, DSR can be thought of as a virtual interface (*dsr0*) residing below the IP layer. Like other groups that have used virtual interfaces to hide mobility from the normal network stack [5], *dsr0* accepts packets from the normal IP stack just as any other interface would, but uses its own mechanisms to arrange for their delivery via the actual physical interfaces.

Packets originated by a node's transport layer enter the IP layer as normal, where a routing table lookup is used to determine which interface they should be sent out. Nodes that use DSR exclusively (e.g., **T1-5** and **E1**) have a default route directing all of their packets out the *dsr0* interface. When a packet is passed to the `dsr_output()` routine, DSR checks the Route Cache for a route. If a route is *not* found, the packet is inserted into the Send Buffer and a Route Discovery is invoked for the packet's destination. Otherwise, if a route to the destination is found in the route cache, the packet is passed down into the `dsr_xmit()` routine. The `dsr_xmit()` code is responsible for delivering the packet to its next hop, and so saves a copy of the packet into the Retransmission Buffer before handing it down to the output queue of the physical interface.

Every 30 ms, a Route Discovery timer inspects the contents of the Send Buffer and, subject to the rate limiting described in the DSR Internet-Draft [2], initiates Route Discovery for any packets found in the Send Buffer. Likewise, a retransmission timer runs once per 50 ms, examining the contents of the Retransmission Buffer and retransmitting packets or generating ROUTE ERRORs as necessary.

Because the radio interfaces we used are inherently broadcast and the CPUs are significantly faster than the link bandwidth, we simplified our implementation by eliminating all need for ARP. All packets sent by `dsr_xmit()` are sent to the MAC broadcast destination address, so all nodes receiving the packet will process it. This actually causes no extra processing overhead at the receiving nodes, since DSR operates the network interface in promiscuous receive mode to implement many of its optimizations [2].

As explained in Section 6.4, we needed to give the packets with DSR routing information priority access to the link. After two initial attempts resulted in over-simplistic designs that accidentally reordered packets, we decided the cleanest solution was to implement a true multi-queue scheme for the interface output queues. Packets to each interface are demultiplexed into the outgoing queues based on the IP Type-of-Service (TOS) bits in each header.

5.3 Incoming Packets

The IP module reads packets from the IP input queue as normal, and, following the IPv6 rules for extension headers, dispatches the packet to the appropriate upper layer module based on the value of the IP Protocol field. Packets with a Protocol field indicating a DSR header is present in the packet are sent to the DSR options processing routines that handle each of the DSR extension headers. This involves adding the route from a ROUTE REPLY into the Route Cache; removing routes from the cache for ROUTE ERRORS; removing packets from the Retransmission Buffer for DSR ACKNOWLEDGMENTS; and forwarding packets based on the Routing Header. Packets that contain transport layer data for the processing node are handed up to the transport layer.

5.4 Interfacing with User-Level Processes

The DSR module communicates with user-level agents on the same node via an *ahsock* (ad hoc network control socket), derived from the BSD concept of a routing socket. The ahsock provided a general purpose conduit for information exchange between all user-level and kernel-level modules on a node concerned with the node's network operation. For example, whenever DSR originates a data packet, it places the node's *location information* into the DSR headers on the packet. This information is used for diagnostic purposes in the testbed, and is comprised by the node's latitude, longitude, heading, and speed. The in-kernel DSR module learns the location information from a user-level process called *GPSd* (Section 6.5.1) which reads the information from the GPS unit and writes it into the ahsock. Similarly, whenever the DSR module processes a packet from another node, it extracts the location information from the headers and sends the information to the ahsock. As a result, processes on the node listening to the ahsock can learn the last known location of the other nodes that are originating packets. The ahsock also proved itself an extremely valuable portal for invoking test code inside the DSR layer and viewing the results.

5.5 Mobile IP

The Mobile IP functionality is split between user space and kernel space, with a small encapsulator and decapsulator module inside the kernel being controlled by the user-level *mobiled* process. *mobiled* uses a collection of *network interface controllers* to monitor the status of each interface, and to gather information on any home agents or foreign agents reachable via the interface. Once it decides which interface is currently the best for communication, it uses the *rtsock* routing socket to manipulate the kernel and routing table state to use that interface.

6 Implementation Features

This section describes some of the notable features of our implementation, namely our scheme for integrating ad hoc networks with the Internet, the adaptive retransmission timer used by the DSR layer, and several logging and support utilities that we found useful when working with the ad hoc network testbed.

6.1 Integration with the Internet

We have extended the mechanisms of Route Discovery and Route Maintenance to support communication between nodes inside the ad hoc network and those outside in the greater Internet. So that each node in the ad hoc network maintains a constant identity as it communicates with nodes inside and outside the network, we require that each node chooses a single IP address, called its *home address*, by which it is known to all other nodes. This notion of a home address is identical to that defined by Mobile IP [22]. As in Mobile IP, each node is configured with its home address and uses this address as the IP source address for all of the packets that it sends.

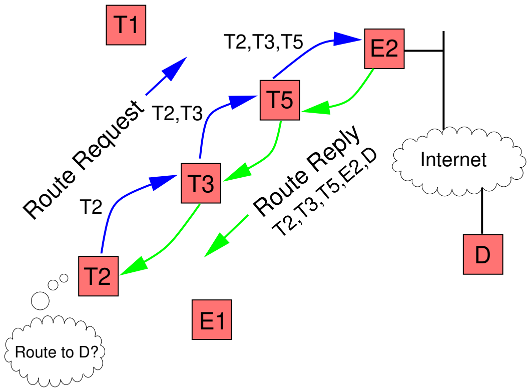


Figure 6 Route Request for a node not in the ad hoc network being answered by the Foreign Agent

Figure 6 illustrates node **T2** inside the ad hoc network discovering a route to a node **D** outside the network. As the ROUTE REQUEST from **T2** targeting **D** propagates, it is eventually received by the gateway node **E2**, which consults its routing table. If it believes **D** is reachable outside the ad hoc network, it sends a *proxy reply* listing itself as the second-to-last node in the route, and marking the packet such that **T2** will recognize it as a proxy reply. If the target node **D** actually is inside the ad hoc network, then node **T2** will receive a ROUTE REPLY from from both **E2** and **D**. Since **T2** can distinguish which replies are proxy replies, it can prefer the direct route when sending packets to **D**. Our method of integrating ad hoc networks with the Internet and several related issues are described in more detail in a separate publication [3].

6.2 Acknowledgment and Retransmission Mechanism

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 Route Maintenance. Each packet a node originates or forwards is retransmitted until an acknowledgment from the next hop is received, or until three transmission attempts have been made, at which point the next hop is declared unreachable and a ROUTE ERROR is sent to the originator of the packet.

We utilize passive acknowledgments [15] 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 the 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. This 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**. Node **A** also requests an explicit acknowledgment from **B** if **B** is the packet's final destination, since in this case, **A** will not have the opportunity to receive a passive acknowledgment from **B**.

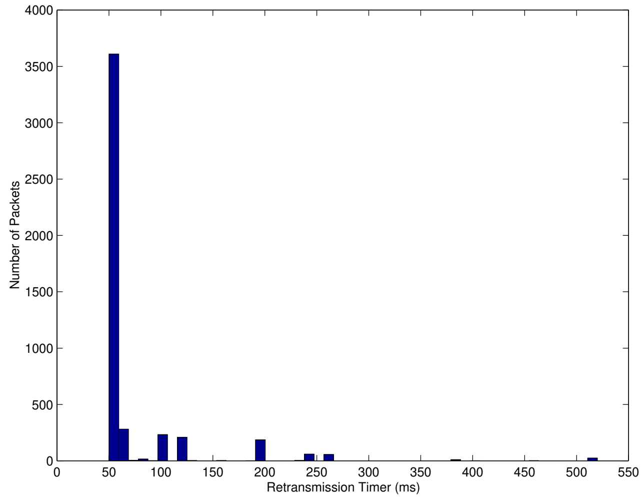


Figure 7 The distribution of values for the DSR retransmission timer over several runs.

For each next hop to which a node has recently attempted to forward packets, it keeps a separate estimate of the round trip time (RTT) between it and the next hop. This RTT is then used to initialize the retransmission timer for each packet sent to the next hop. Our simulation studies [4] of the acknowledgment scheme in IMEP [6] suggested that an adaptive retransmission timer would be needed to accommodate competition for the shared link by other nearby nodes. In keeping with this, we used the TCP RTT estimation algorithm [26] to adapt the RTT for each next hop.

Figure 7 shows the number of times a packet was sent with a given value of the retransmission timer over the course of several runs. Of the 4710 measurements in this particular data set, two values (both 920 ms) are not show in this figure. 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 probably required for good performance if acknowledgments are implemented at a layer above the link layer. IEEE 802.11, for example, does not require an adaptive retransmission timer since the acknowledgment is a scheduled, atomic part of the exchange of a single data packet, and so the time between transmission of a data packet and the receipt of the acknowledgment is not effected by the number of nodes attempting to acquire the media.

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 packet if an identical copy of this packet was found in either its Send Buffer or its Retransmission Buffer. We found that this simple form of duplicate prevention was sufficient, and that maintaining a history of recently seen packets was not necessary.

In order to limit load on the CPU, we set the granularity at which the retransmission timer is serviced to 50 ms. This value was chosen based on observed retransmission timer values and the average one-hop RTT, which is 30 ms.

6.3 Managing the Retransmission Timer

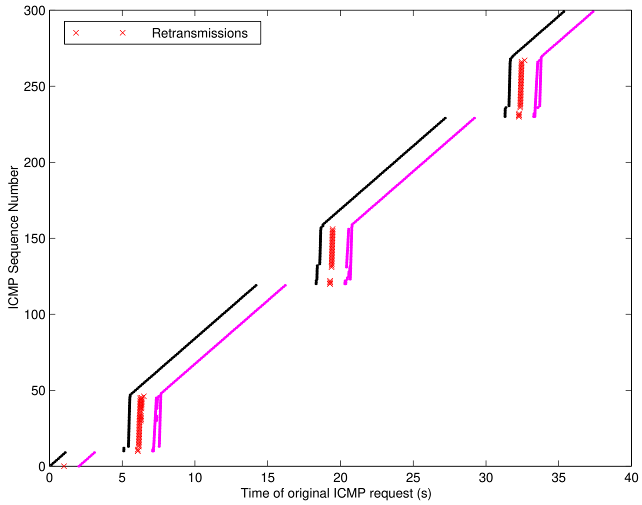
As described in Section 6.2 we implemented a hop-by-hop acknowledgment and retransmission scheme within the DSR layer for the purpose of performing Route Maintenance. However, we found that large numbers of packets are lost or retransmitted needlessly if the retransmission timer does not adapt quickly enough during periods of network congestion. During times of network congestion, the time between when a packet is sent and the acknowledgement received increases due to the need for both packet and acknowledgement to compete for the media.

We found that a simple method of reacting to increasing congestion did not work. If retransmission timer expirations 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.

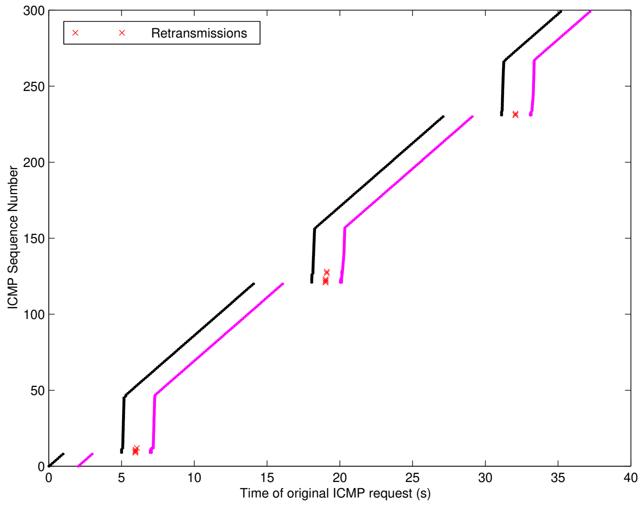
We developed a successful retransmission timer algorithm by including a heuristic estimate of the level of local congestion, so that the retransmission timer could react quickly to changes. One of the simplest ways for a node to measure congestion in the network is to look at the length of *its own*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

The result of a simple experiment demonstrating this idea is shown in Figure 8. This figure represents an experiment in which node **A** sent 300 ICMP Echo Request packets to node **B** at a rate of 10 packets per second. The interface queue at node **A** was alternately disabled (not serviced by the device driver) for three seconds and then enabled for 10 seconds in order to simulate congestion. In Figure 8(a) and (b), the left-most line in the sequence plot represents the time that an Echo Request with the specified sequence number was sent from **A** to **B**. The center line of X's (shifted 1 second for clarity) indicates what ICMP sequence numbers were retransmitted by the DSR layer at node **A**, and the right-most line (shifted 2 seconds for clarity) indicates when node **A** received the corresponding Echo Reply packets from **B** for each ICMP sequence number.

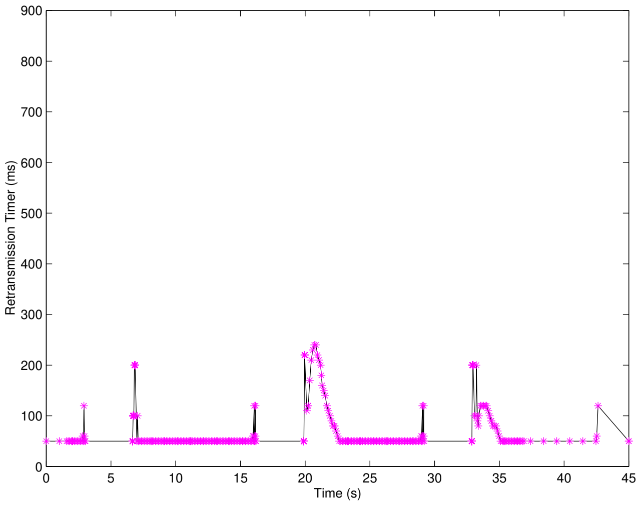
Figure 8(a) shows the ICMP sequence number plot for the case in which no additional heuristics were used to adapt the retransmission timer. Figure 8(b) shows the same plot for the case in which the queue length was taken into account as discussed above. Comparing the two figures, there is a total of 118 retransmissions in the case in which the queue length



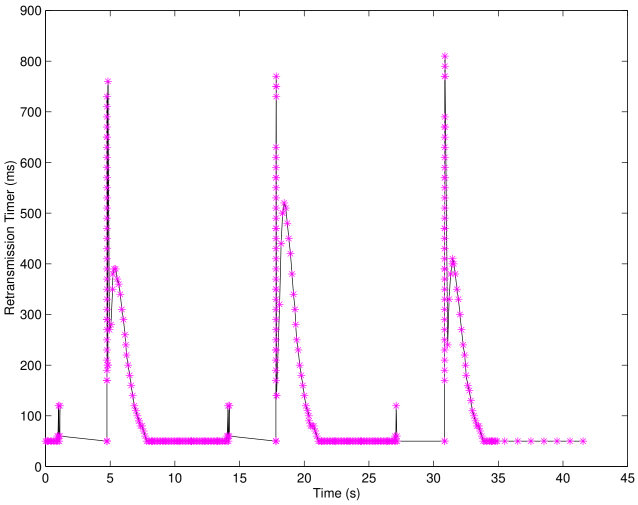
(a) ICMP sequence plot – no heuristics



(b) ICMP sequence plot – queue length heuristic



(c) DSR retransmission timer – no heuristics



(d) DSR retransmission timer – queue length heuristic

Figure 8 Retransmission behavior with and without heuristics to adapt to congestion. The time base of retransmissions and replies in figures (a) and (b) have been shifted for clarity.

was not considered, but only 16 retransmissions occur in the case in which the queue length was used to help adapt the retransmission timer.

Figures 8(c) and (d) show the value of the retransmission timer for a given packet at the time at which it was sent by the DSR layer, in order to illustrate the reason for the difference in the number of retransmissions. When the queue length is not used as a heuristic to set the retransmission timer, the maximum value used for the retransmission timer is 240 ms (Figure 8(c)). When the queue length is accounted for (Figure 8(d)), the retransmission timer backs off much more quickly when the queue begins to fill up, attaining a maximum value of 810 ms. However, as the queue empties, it quickly returns to its normal (measured) value. This behavior results in significantly fewer retransmissions by the DSR layer during periods of transient congestion.

6.4 Prioritizing Packets

It is critical that packets generated by the routing protocol propagate through the network in a timely fashion. For example, ROUTE REQUESTS that are significantly delayed prevent data from being transmitted. Similarly, delaying the transmission of a ROUTE ERROR allows bad state to remain in the network (thereby increasing packet loss), and delays in transmitting network-layer acknowledgments may lead to excessive packet retransmissions that will congest the network.

For this reason, all DSR packets (REQUESTS, REPLIES, ERRORS, and ACKNOWLEDGMENTS) are given higher priority than normal data packets. Additionally, data packets that are being retransmitted by the DSR layer are also treated as high priority packets, since the failure to get such a packet through may result in an upper-layer protocol retransmitting the packet. We noted these competitive retransmissions in our early testing, but did not quantify their impact before changing the priority of retransmissions as described here to compensate.

To enable priority-based scheduling of packets transmissions, we implemented a multi-level queue scheme in which packets are placed in different queues based on their priority. As described in Section 5.2, priorities are assigned to packets using the Type-of-Service (TOS) field in the IP header. Currently, we need only two-levels of priority: low delay and normal priority. The highest priority queue is always completely serviced before any packets from the next-highest priority queue are transmitted. Although the potential for starvation exists with this scheduling algorithm, we have found in practice that starvation does not occur.

6.5 Logging and Support Utilities

Each node in the testbed runs a series of user-level logging and monitoring utilities. In the context of the testbed, the output of these utilities serves as the basis for the post-run analysis of each experiment. In a deployed network, some of this information could also be valuable directly to the end user.

6.5.1 Global Positioning System Information

Each node in the network is outfitted with a GPS receiver and runs a program called *GPSd*, which reads the current position information from the GPS receiver over a serial cable and makes it available to the other processes on the node via a local socket (Section 5.4). Whenever a node originates a packet, the DSR code includes the node's current location in the header of the packet so that each of the packet's recipients will learn the location of the sender. By logging the GPS information, we can analyze the behavior of DSR during a run and can, for example, recreate any run of the testbed in simulation.

As described in Section 4.3, the GPS receivers must be continually supplied with up-to-date corrections in order to obtain the highest degree of accuracy in the GPS position information. This correction information is generated once per second by the GPS reference station located at the field office, and must be delivered to the GPS receivers with minimum delay in order to be useful. Typical GPS deployments use a long-range broadcast radio to distribute the corrections, but in our testbed, we used the ad hoc network itself to distribute the corrections.

The correction information is generated at the reference station in a Trimble proprietary binary format, which is encapsulated into IP packets and transmitted across the point-to-point 2.4 GHz wireless link to **E2**. From **E2**, the corrections are multicast into the ad hoc network by piggybacking the data onto a ROUTE REQUEST targeted at a multicast address [2]. The correction packets propagate hop-by-hop across the ad hoc network, and the GPSd process on each node loads the correction information into the GPS receiver. In our experiments, the routing protocol delivered the correction packets with low enough delay that all GPS receivers typically operated in the highest accuracy RTK mode, where position fixes have accuracy within about 1 cm.

6.5.2 Network Monitoring and Fault Diagnosis

In order to gather real-time statistics on node performance and status (e.g., number of packets forwarded and number of ROUTE REQUESTS sent), each node runs a process called the *Position and Communication Tracking daemon (PCTd).* The purpose of the PCTd is to enable real-time monitoring and visualization of the network as it runs, and to enable a replayed visualization of a run after it concludes. To create a permanent record of the run, PCTd logs the data to a local file, and to facilitate real-time diagnosis, it unicasts the information that it collects over the ad hoc network to a visualizer **V** at the field office (Figure 2).

The visualizer application is written in Tcl/Tk and displays a map of the site showing the current location of all vehicles, based on the GPS information reported from PCTd. Clicking on a node brings up event logs for that node, and allows the user to open strip-chart plots of information, such as error rates and packet forwarding rates, that were recorded at the node.

Figure 9 shows a screen shot of the visualizer application during the replay of a testbed run. The main window shows the location of the nodes, based on the information reported by PCTd. The clock at the top right shows the elapsed time in seconds since the run began. Across the bottom are boxes for each node which show the quality of the position fix provide by GPS (all nodes show Real-Time Kinematic Float in the figure) and the time at which the last PCTd update was received from that node. The three windows down the side show strip-chart recordings of the number of packets per second handled by nodes during the run as a function of elapsed time. Reading from top to bottom, they show:

1. The number of DSR packets per second forwarded by node 6 (labeled **E2** in Figure 2). This indicates the total traffic into and out of the ad hoc network.
2. The number of TCP packets per second received by the transport layer on node 7 (labeled **E1** in Figure 2).
3. The number of UDP packets per second sent by node 3 (labeled **T3** in Figure 2) while the synthetic voice traffic generator was running.

The visualizer serves three major roles in our testbed. First, since the PCTd packets themselves traverse the ad hoc network, simply receiving PCTd updates indicates that the basic network is functioning. Should the network fail, PCTd does log all its information on local disk where it can be recovered after the run. Second, the visualizer has proven crucial for explaining to others what is happening in the network. Since the course is over 700 m long and makes several turns, there is almost no way to see all of it at once. Without a "bird's eye" view of the network, it was hard to make people grasp what they were seeing. Finally, the network is of sufficient complexity that even we as its designers and operators needed

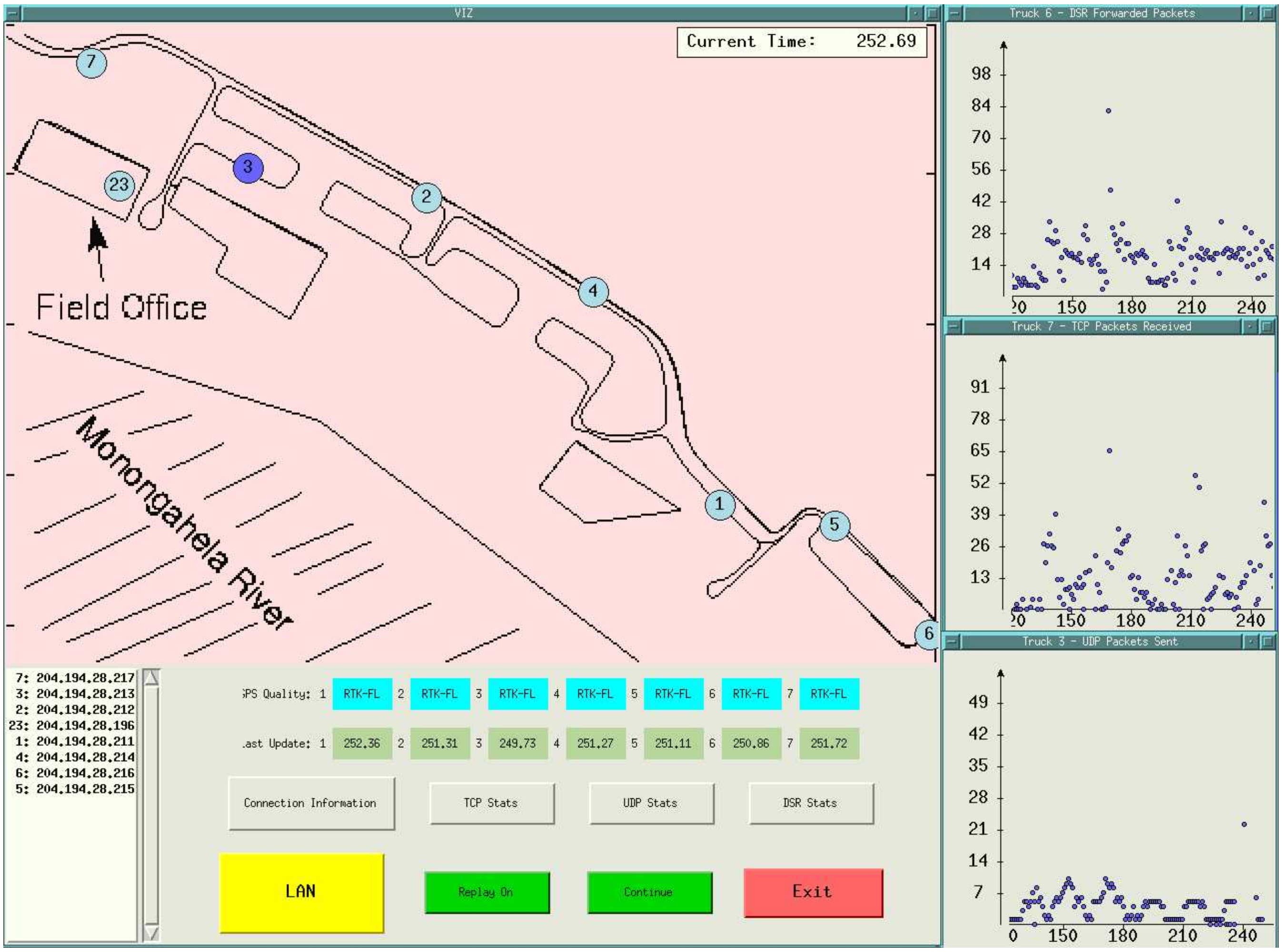


Figure 9 Screen shot of VIZ, the network monitoring and fault diagnosis tool. The main window shows the location of the nodes based on the information reported by PCTd. The three windows down the side show the number of packets per second forwarded by node 6, received by TCP on node 7, and sent by UDP on node 3, respectively.

a network management console in order to look for subtle interactions and diagnose problems on the fly.

6.5.3 `tcpdump` *and Signal Strength*

During a standard run, each node uses `tcpdump` to record to local disk all the packets it received during the run. Post-run analysis of these packet traces is the single best source of information we found for understanding the activity of the network. As a caveat, we did find that even though we used 233 MHz Pentium laptops connected to a link with a maximum bandwidth of 2 Mb/s, when `tcpdump` attempted to capture the entire packet (snaplen = link MTU), the bursts of disk activity for large streams of received packets were enough to cause the machines to delay sending their acknowledgment packets past the expiration of the retransmission timer. The resulting retransmissions more than *halved* the goodput we measured for TCP and CBR transfers — indeed, during several experiments that we ran in the lab, TCP goodput dropped from 0.9 Mb/s to 0.3 Mb/s when `tcpdump` was run with a snaplen of 2000 bytes. We found that saving only the first 200 bytes of data from each packet did not adversely affect TCP's goodput.

For each packet received, the WaveLAN-I cards report the "signal strength" and "signal quality" with which they received the packet — information which is useful for characterizing the quality of the wireless link. Since this information is associated with each received packet, we originally found that the most convenient way to make this information available to our monitoring applications was to append the signal strength information to each incoming packet before the WaveLAN device driver tapped the packet off to the Berkeley Packet Filter (BPF). It was the need to record the metrics at the end of each packet that originally motivated us to have `tcpdump` record the entire packet. Instead, we were able to capture the signal strength information without overloading the machines by extending the BPF header (`struct bpf_hdr`) with a generic interface-metrics field so that the signal information could be tapped off with each received packet, regardless of the snaplen specified.

6.5.4 Hardening the Network Tools

When working with standard utilities such as *Netperf* [14] and with our own *PCTd*, we found that they frequently exited, crashed, or aborted. Specifically, many network utilities terminate upon receiving an error from system calls such as `write` or `sendto`. However, in the the dynamic, unpredictable environment of an ad hoc network where temporary partitions are unavoidable, we found that these programs needed to be altered so that the receipt of an ICMP Destination Unreachable message did not result in a fatal error. We generally handled these errors by putting the process to sleep for a few seconds, clearing the error on the socket, and then allowing it to retry its previous operation.

6.5.5 Per-Packet State Tracing

In order to understand the packet traces we recorded, it was useful to have access to the internal state variables of the protocols. For our TCP traffic generator, we used the program *DBS* [19], which both creates an offered load and utilizes a small kernel module that records the contents of the TCP protocol control block every time a packet is sent or received. DBS then saves the collected control blocks to a local file.

Likewise, to verify parts of the DSR implementation, such as the retransmission timers, we implemented a logging device akin to `syslog` called `/dev/kdsr` to which DSR wrote its internal state variables every time it sent a packet. A user-level program read these binary records from the kernel and logged the variables to disk for later analysis. `syslog` was not appropriate for this task, because this logging had to be very efficient to avoid disrupting the system. We believe converting log records to printable strings on critical code paths within the kernel would have been too costly an operation.

6.5.6 Traffic Generation Tools

In order to obtain good experimental control over the load offered to the network, we developed an Application Traffic Generator (ATG) program capable of performing three types of data transfers with any other machine in the network also running the ATG. The ATG currently supports synthetic voice calls, bulk data transfer, and location dependent transfers. Voice calls are synthesized by a stream of 1000-byte UDP packets being sent from caller to callee for 10 seconds at a rate of 8 packets per second. The caller then pauses for 10 seconds while the callee sends the caller a similar data stream for 10 seconds. This pattern then repeats for the duration of the call to simulate the back and forth nature of speach. Bulk data transfers consist of a TCP connection transferring a file from one machine to another (i.e., FTP). Location dependent transfers make use of the GPS information available on the nodes to begin a transfer whenever the sending node enters an area specified by GPS coordinates and end the transfer when leaving the area.

The pattern, rate, and size of the connections are specified to the ATG running on each machine by a configuration file. Table I shows the load carried by the testbed network during full runs. It includes: each node making one voice call

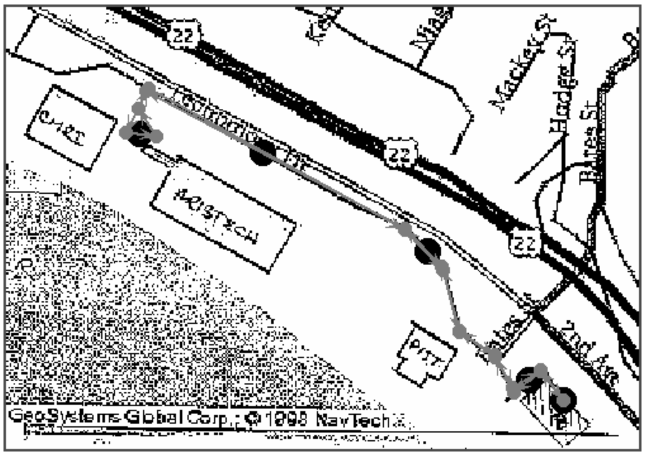


Figure 10 The map display of `ad-hockey` showing the positions and trajectory of the nodes. The trajectory can be changed by click-and-drag on the grey knots.

to every other node once per hour; each node transferring a data file to every other node once per hour; and each moving node (**T1-5**) making a location dependent transfer to **E1** when within 150 m of **E1**. The table also shows the load created by the "differential GPS corrections" (Section 6.5.1) and the situational awareness (PCTd) information (Section 6.5.2) carried on the network.

7 Preliminary Testing and Course Evaluation

7.1 Initial Node Testing

The initial testing of our implementation of DSR was hampered by the fact that we intended to use radios with a range of about 250 m, and testing the implementation meant repeatedly and repeatably moving the nodes in and out of range of each other. Not having the services of the school track team available for this, physically moving the nodes in order to cause network topology changes while controllably debugging the implementation would be an intractable problem. To enable us to test our implementation quickly on a wide range of topologies and movement patterns, we developed a simple ad hoc network emulation system.

We placed all of the physical laptops running the actual DSR code together in our lab, each connected to their actual radio. Since each machine was physically within direct range of all the others, we could emulate any desired topology by simply preventing the network stack of each machine from processing the packets that it would not have been able to receive had the nodes actually been deployed in the field and separated by varying distances. To achieve this, we implemented a packet killer called the *macfilter* between the physical interface and the network layer input queue. The macfilter checks the MAC source address of each received packet against a list of prohibited addresses. Packets whose source addresses are found on the list are silently dropped. Emulating the motion of the nodes then reduces to the problem of loading the macfilter with the proper list of prohibited addresses at appropriate times.

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 broadcast	UDP	150 bytes
PCTd	1 pkt/sec/node unicast	UDP	228 bytes

We control the macfilter packet-killer with a *trace file* consisting of the MAC source address lists and the times at which these lists should be loaded into the macfilter. The trace files are created by using our `ad-hockey` graphical scenario creation tool to draw the paths the nodes should move along (Figure 10). The `ad-hockey` tool then generates the trace files using a trivial propagation model where any nodes falling within a fixed radio range of each other can communicate, and any falling outside the range cannot. As the emulation runs, `ad-hockey` displays the "positions" of the nodes as they "move", allowing the testers to correlate node positions with protocol behaviors.

Our emulation system of a packet-killer controlled by a trace file is similar in concept to the Network Trace Replay system [21], but neither it nor any of the other existing emulation systems available at the time had the expressive power to emulate a multi-hop ad hoc network. At a minimum, the packet-killer must be able to express the notion that a node **A** can receive packets from **B** while simultaneously and independently being unable to receive packets from **C**. To be convenient, the emulation system must also provide tools for easily generating the packet-killer control files. For testing ad hoc networks, where the topology changes are driven by movement, we found having a tool that allowed us to draw the "motions" of the nodes extremely useful.

We do not claim that performance measurements made on the emulated network created by macfilter in any way approximate the measurements of a deployed network, and, in fact, we have verified that results obtained in the lab are not comparable to those collected in the field. Our packet-killer and trace generator do not emulate the variability of the outdoor wireless environment, and all the nodes are in the same collision domain, so there are no hidden terminal problems.

Nevertheless, the macfilter was a critically important tool during the early stages of developing and debugging the ad hoc networking code, as we could exercise all aspects of the protocol from within our lab without having to spend countless hours running around campus to create physical topology change. Using the emulation system, we created scenarios to individually test all the protocol's reactions to topology change, including Route Discovery, Route Maintenance, and the retransmission timers. The macfilter also allowed us to perform regression testing on the implementation, and to find bugs that only appeared after tens of hours of running, whereas we could not possibly ask our human drivers to keep driving that long.

Since developing the macfilter as a protocol development tool, we have begun working to develop and validate systems that support true ad hoc network emulation [18, 9] which *can* be used for the accurate performance analysis of protocols and systems running on top of an ad hoc network.

7.2 Characterizing the Course and Equipment

As part of conducting an initial survey of the site, we found it particularly helpful to obtain a rough characterization of the site's propagation environment. We had two cars drive the course at about 30 Km/hr (20 miles per hour), one following the other, with the trailing car transmitting 1024-byte packets to the lead car 10 times per second. For each experiment, the cars made three laps of the course, a total driving time of about 660 seconds. We conducted experiments with the cars separated by 10, 25, and 50 seconds, which on the straightaways is a distance separation of approximately 90 m, 220 m, and 440 m, respectively.

During the experiments, each node recorded all packets heard by its radio using `tcpdump`. We then used a packet trace differencing tool that we developed to find the packets sent by the trailing car that were not received by the lead car. For all experiments, we found that the transmission losses occurred in regular bursts synchronized with the 220-second lap time. For example, Figure 11 depicts the losses as function of time for the 25-second separation experiment, with a line drawn at 1% of total losses.

When plotted on a map of the site, the location of the worst propagation areas turned out to be surprising. Figure 12 shows a '$\times$' mark and a '+' mark for each time period in which the loss was greater than the 1% line in Figure 11. The '$\times$' mark depicts the point where the trailing car was located when it transmitted the packets that were lost in transmission, and the '+' mark depicts the point where the lead car was located when it failed to receive the packets. All of the loss bursts occurred while the nodes were on *the straightest part of the course with clear line-of-sight to each other*. There is no elevation change along that portion of the course, and the parking lots along the south side of the road were empty during the test.

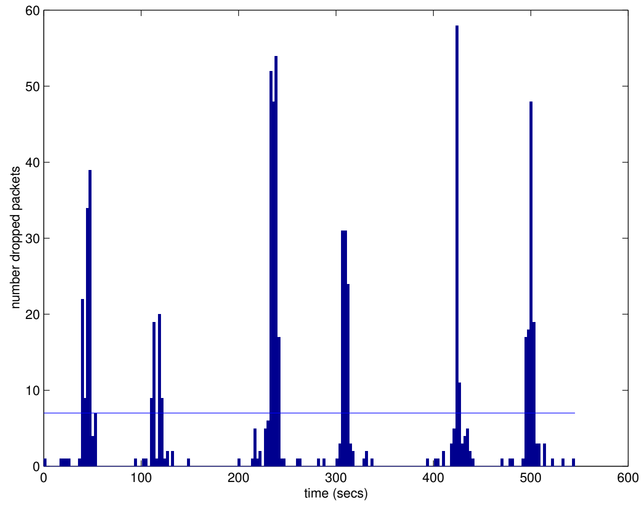


Figure 11 Times at which packets were lost during a run. Horizontal line indicates 1% of total number of packets. Nodes maintained 25 s separation (220 m).

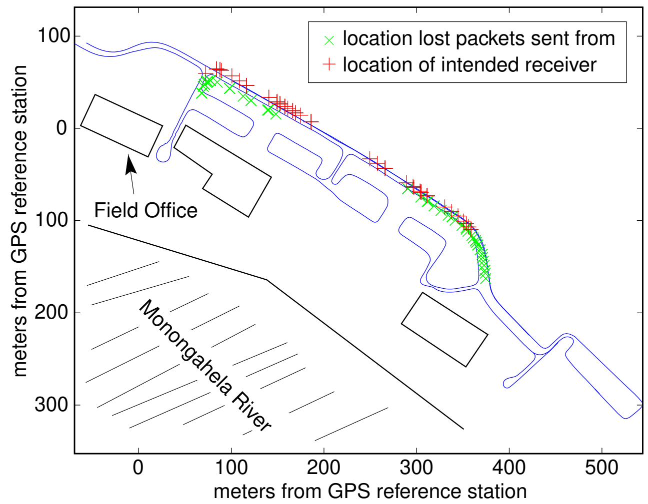


Figure 12 The location of the sending node and receiving node when more than 1% of packet loss occurred. Nodes maintained 25 s separation (220 m).

To rule out the possibility that the drivers were merely speeding up on the straightaway and so increasing the distance between the nodes, we reran the experiment with a separation of 10 seconds (90 m) between the nodes. The results, depicted in Figure 13, show that even with a spacing between the cars of much less than half the radios' nominal range of 250 m, there are still significant loss bursts in front of the field office building. Our current hypothesis is that the radios are suffering from multipath reflection off of the flat fronts of the buildings, in particular the large building next to the field office.

8 Ping Test Results

As an initial end-to-end measurement of the ad hoc network's ability to route packets over multiple hops between the nodes,

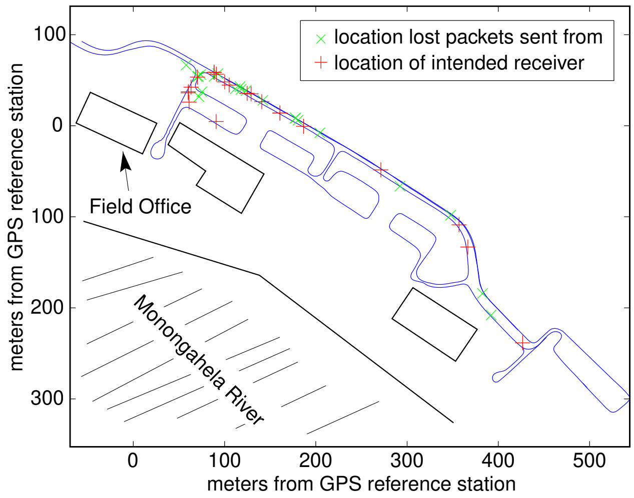


Figure 13 The location of the sending node and receiving node when more than 1% of packet loss occurred. Nodes maintained 10 s separation (90 m).

we conducted a test in which the 5 moving nodes **T1**-**T5** drove the course shown in Figure 3 at a speed of 30 Km/hr (20 miles per hour) while maintaining a separation of approximately 200 m.

E1 then offered a traffic load to the network, consisting solely of 64-byte ICMP ECHO REQUEST packets directed at stationary node **E2**, sent every 300 ms for approximately 1000 seconds. Since **E1** and **E2** are not within wireless transmission range of each other and since the cars continue to loop between these points, the sequence of hops taken by each packet was variable. Including the IP header (20 bytes), the WaveLAN header (16 bytes), and the GPS information piggybacked on each packet (44 bytes), the minimum size of these packets was 144 bytes. As explained in the following sections, these ping tests served a variety of purposes.

8.1 Quality of Communication

During this test, **E1** originated 3343 unique ICMP ECHO REQUEST packets. **E2** returned 3013 ICMP ECHO REPLY packets for an overall end-to-end loss rate of 10%. This number gives us a rough approximation of the type of loss rate that applications will experience when trying to operating within the ad hoc network.

Since 10% is a high loss rate for end-systems to cope with, we examined the network dynamics of this test more carefully to determine what caused the losses. Due to the setup of the test, **E1** alternated among using each of the moving nodes as the first hop on its route to **E2**. Table II shows the raw loss rate over a single hop from node **E1** to each of the moving nodes, excluding packet retransmissions by the DSR layer. These numbers characterize the error rate of the wireless link and show what the per-link loss rate would be in the *absence* of retransmissions by DSR.

In contrast, Table III shows the loss rate of the links between **E1** and each of the moving nodes when including DSR

Table II The packet loss rate experience by node **E1** to each first-hop node when counting the number of unique ECHO REQUESTs transmitted by **E1**. The *packets received* column does not include retransmissions by the DSR layer.

	Packets Sent	Packets Recv	Loss Rate
T1	730	635	13.01%
T2	525	459	12.57%
T3	658	604	8.21%
T4	472	414	12.29%
T5	947	844	10.88%
Total	3332	2956	11.28%

Table III The packet loss rate experience by node **E1** to each first-hop node when counting the number of unique ECHO REQUESTs transmitted by **E1**. The *packets received* column includes retransmissions by the DSR layer.

	Packets Sent	Packets Recv	Loss Rate
T1	730	681	6.71%
T2	525	489	6.86%
T3	658	631	4.10%
T4	472	448	5.08%
T5	947	907	4.22%
Total	3332	3156	5.28%

retransmissions. The significant reduction in loss rate over each link shows that the DSR retransmission scheme is serving its purpose.

8.2 Verifying the DSR Retransmission Algorithm

In addition to verifying that DSR retransmissions were improving the per link loss rate perceived by applications, we also needed to verify that DSR was not retransmitting excessively and thereby burdening the network. During this 1000-second ping test, **E1** transmitted 4401 ECHO REQUEST packets. As mentioned in Section 8.1, only 3343 of these packets had unique sequence numbers, indicating that 1058 packets (24% of the total number of transmissions) were actually retransmissions or duplicates generated by the DSR layer.

Of the 1058 retransmitted or duplicate packets, 185 were found to be packets removed from the Retransmission Buffer and recycled by the DSR layer because the packet's first-hop destination was found to be unreachable.

In order to explain the remaining retransmissions, we attempted to verify that for each first hop node (**T1**-**T5**), the number of retransmissions made by **E1** satisfied the following criteria, which captures the notion that packets must be retransmitted until they are acknowledged:

$$Rxmt \approx Orig - ACK_{passive} + Rxmt_{lost} + ACK_{lost}$$

where $Rxmt$ is the total number of retransmissions made by **E1**, $Orig$ is the number of unique packets originated by **E1**, $ACK_{passive}$ is the number of passive acknowledgments received by **E1**, $Rxmt_{lost}$ is the number of retransmissions made by **E1** that were not received by the first-hop node, and

Table IV The the number of ECHO REQUESTs transmitted by node **E1** to each first-hop node and the number of passive acknowledgments that **E1** received from each first-hop node. The expected number of retransmissions (the difference of column 1 and column 2) is shown in the third column. These numbers do not include retransmissions by the DSR layer.

	Packets Sent	Passive Acks	Expected Rxmts
T1	730	613	117
T2	525	445	80
T3	658	601	57
T4	472	410	62
T5	947	842	105
Total	3332	2911	421

ACK_{lost} is the number of explicit acknowledgments transmitted by the first-hop node that were not received by **E1**. For purposes of analysis, we rewrite the equation as:

$$Rxmt - Rxmt_{lost} - ACK_{lost} \approx Orig - ACK_{passive}$$

Table IV evaluates the right half of this equation and shows how many ECHO REQUEST packets were originated by **E1** when using each of the moving nodes as the first hop and how many passive acknowledgments were received as a result of the first-hop node forwarding the packet. The difference of those two values is the number of retransmissions that we would expect **E1** to attempt to each first-hop node. The total number of expected retransmissions is 421.

Table V evaluates the left half of the equation, showing the actual number of retransmissions made by **E1** to each of the moving nodes and the number of these retransmissions that were lost. Subtracting these numbers yields the number of retransmissions received by the first hop node. Each retransmission must be explicitly acknowledged by the first hop node, and the loss of the explicit acknowledgement will cause **E1** to retransmit the packet again. Subtracting the lost explicit acknowledgements yields the total number of successful retransmissions made by the network. Summing over all first hops shows there were a total of 345 successful retransmissions.

Using this rough estimation technique, we see that the number of expected retransmissions shown in Table IV is greater than the number of successful retransmissions shown in Table V. This verifies that DSR is not retransmitting spuriously. The numbers are not exactly equal due to packets that are discarded after being retransmitted the maximum allowed number of times.

8.3 Variability in the Environment

Occasionally during our testing, we found that **E1** could communicate directly to **E2**, a distance of 671 m. Since the 900 MHz WaveLAN radios used in this testbed are specified by Lucent to have a range of about 250 m, this indicates just how variable electro-magnetic propagation can be. During these ping tests, **E1** attempted to send 367 ECHO REQUESTS directly to **E2**. 192 of these packets were originations (not

Table V The difference in the number of retransmissions made by **E1** to each first-hop node and the number of those retransmissions that were lost is the number of retransmissions that got through. Subtracting the number of the explicit ACKs lost yields the number of successful retransmissions.

	Rxmts	Rxmts Lost	Acks Lost	Successful
T1	254	94	17	143
T2	102	63	6	33
T3	75	35	2	38
T4	82	34	2	46
T5	189	81	23	85
Total	702	307	50	345

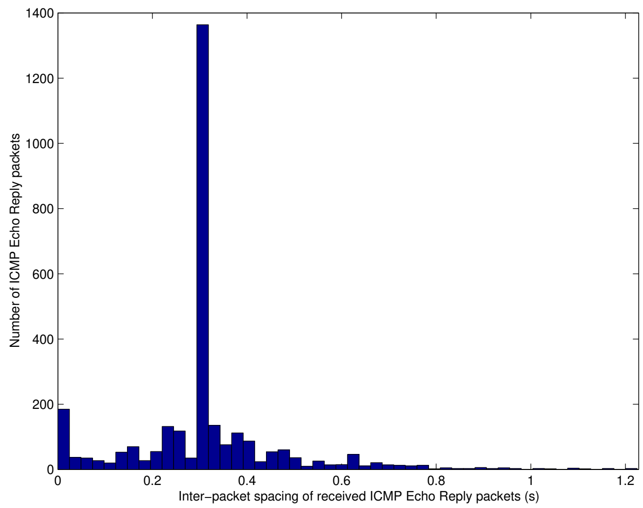


Figure 14 Inter-packet spacing of received ICMP Echo Reply packets when Echo Requests were sent every 300 ms.

retransmissions), and of these 192 originations 149 unique replies were returned from **E2**, giving a loss rate of 22.4%.

8.4 Inter-packet Spacing of ECHO REPLY Packets

During the ICMP test, ECHO REQUEST packets were sent once per 300 ms, so we would expect the inter-packet spacing of ECHO REPLYs to be close to 300 ms as well. As noted above, 3013 ECHO REPLY packets were returned from **E2** to **E1**. Figure 14 shows the inter-packet spacing for the 99th percentile of these packets; 30 measurements were discarded as outliers. The mean inter-packet spacing is 314 ms with a standard deviation of 0.026 ms.

8.5 Route Length

During the ping test, end-system **E1** received 3010 ECHO REPLY packets that we were able to match to specific ECHO REQUESTs. Table VI shows the distribution of route lengths used for both REQUESTs (from **E1** to **E2**) and REPLYs (from **E2** to **E1**). Slightly over 90% of the routes used between E1 and E2 were two- and three-hop routes. Occasionally, a direct route is discovered between the two end-systems, but this route receives little use because of its poor quality.

Table VI The distribution of route lengths used during the ICMP tests. The ECHO REQUEST packets traversed routes from **E1** to **E2** and the ECHO REPLY packets traversed routes from **E2** to **E1**.

	Route Length			
	1	2	3	4
ICMP Echo Request	57	2405	547	1
ICMP Echo Reply	188	2395	422	5

9 TCP Test Results

In addition to the ping tests reported in Section 8, we ran several experiments to characterize how well TCP [23] performs in our ad hoc network testbed. We first ran each experiment in the lab, using the macfilter tool (Section 7.1) to create a multi-hop environment. We then repeated the experiment outside in the actual ad hoc network. The lab measurements serve as a useful benchmark by giving us a best-case approximation to which we can compare the data collected the the field.

9.1 Single-Hop TCP Experiments

Our first set of experiments were intended to establish a baseline TCP performance and verify that the implementation of DSR itself did not degrade TCP's performance. Inside our lab, we set up two DSR nodes and ran TCP benchmarks using DBS across the one-hop link between them. In the single-hop case, passive acknowledgments cannot be used and so a DSR ACKNOWLEDGMENT is transmitted in response to each data packet.

Our first experiment consisted of performing 5 data transfers of 1 MB each in the lab environment. Over these 5 transfers, TCP averaged 0.86 Mb/s (104 KB/s) with a standard deviation of 0.018 Mb/s. This number is less than half of the theoretical link capacity of 2 Mb/s, but coincides closely with the 0.9 Mb/s value typically observed when DSR is not used at all. Due to a choice of MAC protocol parameters inside the WaveLAN PCMCIA cards, node-to-node communication does not proceed as quickly as basestation-to-node communication (which typically sees a throughput of approximately 1.27 MB/s (155 KB/s)).

We then moved both nodes into vehicles in the testbed and separated the nodes by a distance of approximately 250 m. Before starting the TCP test, we verified that the quality of the wireless link between the two nodes was sufficient to allow the nodes to successfully flood ping each other with 1024-byte packets. The nodes then remained stationary for the remainder of the test.

The outdoor test consisted of 5 transfers of 1 MB each and 5 transfers of 5 MB each. The 1 MB transfers had an average throughput of 0.81 Mb/s with a standard deviation of 0.022 Mb/s, while the 5 MB transfers averaged 0.73 Mb/s with a standard deviation of 0.157 Mb/s. The significant decline in average throughput for the 5 MB transfers results from a single outlier with a throughput of 0.4582 Mb/s. The average throughput of the other four transfers was 0.80 Mb/s.

The reasonable and consistent values of our one-hop TCP results assured us that our implementations of DSR's Route Discovery and Route Maintenance mechanisms were performing correctly and not inhibiting TCP's performance.

9.2 Two-Hop TCP Experiments

As with the one-hop TCP experiments, we began the two-hop experiment by collecting data in the lab. Over 5 transfers of 1 MB each, TCP averaged 0.50 Mb/s (61 KB/s) with a standard deviation of 0.079 Mb/s. The variability is explained by the fact that one transfer achieved only 0.36 Mb/s, while all of the others achieved between 0.50 and 0.55 Mb/s.

In order to correctly position the nodes for the outdoor experiment, two cars were driven in opposite directions and positioned as far from an intermediate node as possible, while still allowing both of the end nodes to successfully flood ping the intermediate node with 1024-byte packets. Once positioned, the nodes remained stationary for the remainder of the test.

This created 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 Discovery attempts to restore connectivity. As described in Section 3, Route Discovery consists of a node sending a non-propagating ROUTE REQUEST followed by a propagating ROUTE REQUEST if no ROUTE REPLY is received within 30 ms.

The 1 MB data transfers, which were set up to last for a maximum of 50 seconds, timed out in some of the cases 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 average data rate for the two-hop outdoor scenario was 0.12 Mb/s (14.65 KB/s) with a standard deviation of 0.025 Mb/s, only 25% of the throughput measured in the lab.

The time-sequence number plot from one such two-hop connection in the testbed is depicted in Figure 15. Sequence numbers marked with a small dot were transmitted using a two-hop route through the intermediate node, while sequence numbers marked with the '$\times$' were transmitted directly between the endpoints. The dashed vertical lines in the figure indicate when the TCP source 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.

For the purpose of discussion, let node **A** be the TCP source, **B** the intermediate node, and **C** the TCP sink. Figure 15 shows the TCP connection making very good progress and using almost exclusively a two-hop route for the first 9 seconds of the connection. 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, but 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

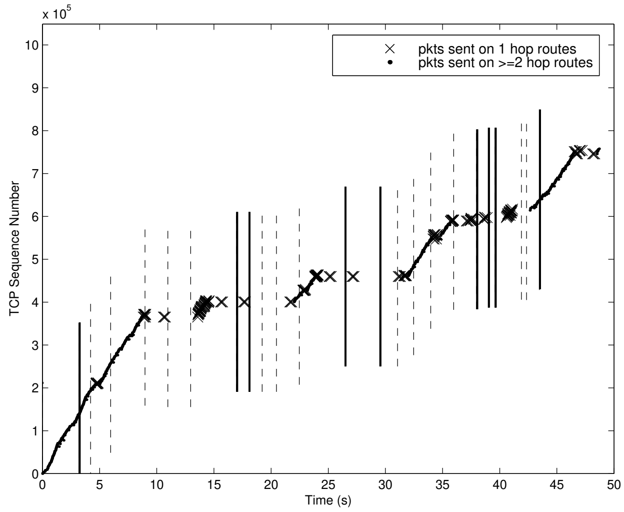


Figure 15 A TCP sequence number plot for a 1 MB transfer over a two-hop route. The vertical lines indicate the times at which the TCP source initiated Route Discovery; the dashed lines indicate the times at which only a non-propagating ROUTE REQUEST was transmitted and the solid lines indicate both a non-propagating and a propagating ROUTE REQUEST .

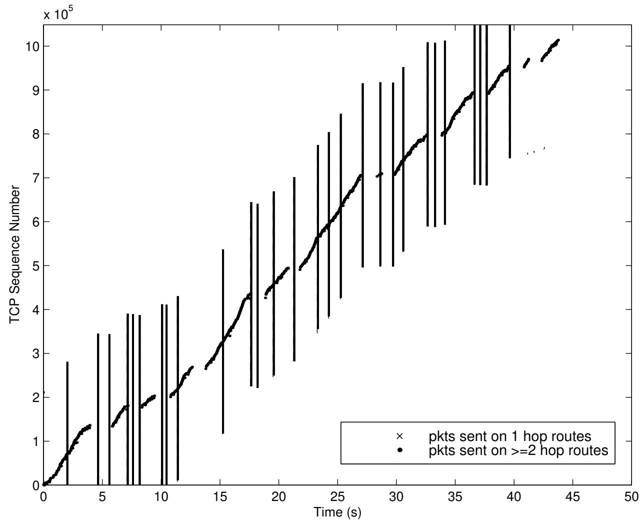


Figure 16 A TCP sequence number plot for a 1 MB transfer over a two-hop route when the *macfilter* utility was used on both the source and destination nodes to prevent the use of single-hop routes. The vertical lines indicate the times at which the TCP source initiated Route Discovery.

the two-hop route through the intermediate node. By this time, TCP has backed off and the next packet is not offered 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 it 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 oftentimes 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) [7] 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 the `macfilter` to eliminate the discovery of 1-hop routes. Figure 16 shows the time-sequence plot for a 1 MB transfer in the field using this "perfect routing." The flat plateaus are missing, and the throughput is 30% higher. 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 certain period of time, but would prevent DSR from repeatedly attempting to use a poor quality link. The drawback of this solution is the difficulty of picking a reasonable timeout value.

A second and more favorable approach would be to use the GPS information propagated by each node to model the position of nodes. 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 that successful communication can occur over a particular route. We intend to experiment with all three approaches over the coming months to determine which approach is most suitable.

10 General Lessons Learned

We are continuing to experiment with the testbed — refining the protocols, the procedures, and the data analysis tools based on our ongoing experience. We believe that building testbeds like ours is a critical step towards a widespread deployment of ad hoc networks. It has already taught us much about the challenges faced by such networks, though this paper focuses on the specific lessons we learned while building the testbed. In summary, we briefly review some of the key issues we discovered.

Obtaining route diversity is interesting, but expensive. As described in Section 4, the core ad hoc network in our testbed is comprised of 7 nodes: 2 stationary end-systems and 5 moving nodes that move back and forth along a road between the 2 end-systems. This scenario works well given the limited number of vehicles, but creates an environment in which there is essentially no route diversity. In other words, at any point in time, there exists only a single route between any two nodes. This linear arrangement of the nodes also increases the hidden terminal problem.

A less linear configuration would allow route diversity, but would increase the partitioning of the network unless more nodes were added. Similarly, the drawback of choosing radios with a short range is that the network is easily partitioned by only tens of meters variations in the relative positions of the vehicles. However, since a goal of the testbed was to challenge the protocols in a rigorous environment, the lack of route diversity has contributed to understanding the protocols' behaviors at one edge of the design space. As more nodes become available, we can then evaluate the scaling and routing issues at other points of the design space.

Adaptive retransmission timers are a necessary complexity. Our experience makes clear that local retransmission algorithms will be a critical part of any multi-hop ad hoc network. Further, if the retransmission algorithm is implemented above the link-layer, it must be adaptive in order to accommodate network congestion and periods of high contention for the wireless channel. We did find that passive acknowledgments were successful in our environment, even though we did not also implement the transmission pacing advocated by the designers of the DARPA Packet Radio Network [15].

Multi-level priority queues are worth implementing. Delivering routing protocol control packets as rapidly as possible is important for good end-to-end performance, and this implies that packets with routing implications should be scheduled for transmission ahead of user's data packets. Our initial implementation of a priority scheme simply prepended all high-priority packets onto the network interface transmission queue. However, this prepending caused reordering of both the stream of control packets and the stream of user data packets, since retransmitted data packets are also considered high priority. We found the most elegant solution was to implement a multi-level queueing scheme, with one queue for each priority at each interface. While we use a trivial scheduling algorithm, more sophisticated ones could potentially be valuable.

Personnel management is nontrivial. A significant limitation on the number of nodes in our testbed was the availability of drivers to operate them. Although groups of three or four people could conduct useful experiments, a full run with all 8 nodes required 7 or more participants for a period of 3 to 4 hours. Since most of the drivers we recruited did not have an overall understanding of the system and project (and several were not even associated with the project), each of the runs started with a "mission briefing" to explain the goal of the experiment and the tasks each node had to perform. Although not described in this paper, we also developed many UNIX shell scripts and procedures to automate as many functions of the runs as possible, in particular the data logging, since several early runs had to be repeated because drivers did not properly configure and start the processes on their nodes. For safety reasons, each experiment had to be controlled autonomously or remotely once the cars started to move, since drivers could not read the screen or type while driving.

In-lab testing of protocols is critical for success. The time we spent developing the macfilter system for emulating ad hoc networks was paid back many times over. It enabled experimentation with multi-hop topologies in the lab, which then served as a useful baseline for comparing with field measurements. Its greatest contribution, in fact, was in debugging and regression testing the implementations to harden them before exposing them to the very unpredictable real world. The macfilter code itself took only a full day to write and test, and it proved easy to integrate with the existing `ad-hockey` tool for scenario generation.

Monitoring and data analysis are improved with GPS information. Accurately knowing the position of each node during a run enabled many useful operations including real-time diagnosis of network behavior and bird's-eye visualization of the entire site using our network visualizer. GPS information also enabled post-run analysis of the performance data that separated the effects of wireless propagation (packet losses due to transmission errors) from routing protocol induced behavior (packet losses due to routing errors).

Wireless propagation is not what you would expect. We found that some of the areas of the site that we expected to have the best propagation, the straight flat areas with direct line-of-site connectivity, in fact had the worst wireless error rates. Conducting an initial survey of the site to identify these areas was useful, even though we did not alter the node movement to avoid the areas, because this knowledge of the site prevented us from unduly wasting time looking for errors in the routing protocol to explain the poor performance in these areas.

We also found that the real world propagation environment will deliver packets between two widely separated nodes with much greater frequency that we expected. Our performance analysis demonstrates that this is a significant challenge that all ad hoc network routing protocols will need to address.

Bystanders will think you are crazy. As an only partially humorous suggestion to anyone building a testbed in an urban area, it will be well worth your time to make friends early with the local police and security guards. They will visit you on a regular basis to find out what your slow moving cars covered in antennas are doing. We were aided considerably by the fact that our site was located in a research park, but it is still worth preparing a list of stock answers. For example, "No, we're not using the radios to conduct industrial espionage," "Our computers have a different purpose from the one in your police car," and, "No, I haven't seen that episode of the X-Files."

11 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. We have characterized the environment of the testbed site, evaluated some basic performance properties of the network and its links, and discovered practical challenges that routing protocols in this environment will need to overcome.

This paper serves to document our experiences designing and building the testbed, in the hopes that the challenges we faced and our solutions will ease the path of future groups attempting to building such testbeds. We are presently carrying out more experiments to characterize and improve the performance of our ad hoc network protocol. We are experimenting with the network's behavior under different levels of traffic load, including audio and video streams, and designing protocol enhancements to provide these streams with quality of service promises. We also intend to use the testbed for evaluating adaptive end-system techniques, including techniques for TCP adaptation, and for application layer adaptation via the Odyssey system [20].

12 Acknowledgements

The CMU 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] David A. Beyer. Accomplishments of the DARPA Survivable Adaptive Networks SURAN Program. In *Proceedings of the IEEE MILCOM Conference*, 1990.

[2] 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-01.txt, December 1998. Work in progress.

[3] Josh Broch, David A. Maltz, and David B. Johnson. Supporting Hierarchy and Heterogeneous Interfaces in Multi-Hop Wireless Ad Hoc Networks. In *Proceedings of the Workshop on Mobile Computing held in conjunction with the International Symposium on Parallel Architectures, Algorithms, and Networks*, Perth, Australia, June 1999. To appear.

[4] Josh Broch, David A. Maltz, David B. Johnson, Yih-chun Hu, and Jorjeta Jetcheva. A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. In *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 85–97, Dallax, TX, October 1998.

[5] Stuart Cheshire and Mary Baker. Internet Mobility 4x4. In *Proceedings of the SIGCOMM '96*, pages 318–329, August 1996.

[6] M. S. Corson, S. Papademetriou, P. Papadopoulos, V. Park, and A. Qayyum. An Internet MANET Encapsulation Protocol (IMEP) Specification. Internet-Draft, draft-ietf-manet-imep-spec-01.txt, August 1998. Work in progress.

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

[8] Francis Dupont. INRIA IPv6 Distribution. ftp://ftp.inria.fr/network/ipv6/.

[9] Kevin Fall. Network Emulation in the VINT/ns Simulator. In *Proceedings of the Fourth IEEE Symposium on Computers and Communications (ISCC'99)*, July 1999.

[10] J.J. Garcia-Luna-Aceves, C.L. Fullmer, E. Madruga, D. Beyer, and T. Frivold. Wireless Internet Gateways (WINGS). In *Proceedings of IEEE MILCOM'97*, November 1997.

[11] IEEE Computer Society LAN MAN Standards Committee. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std 802.11-1997. The Institute of Electrical and Electronics Engineers, New York, New York, 1997.

[12] 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.

[13] 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.

[14] Rick Jones. The Netperf Homepage. http://www.netperf.org/.

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

[16] 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.

[17] Barry M. Leiner, Robert J. Ruth, and Ambatipudi R. Sastry. Goals and Challenges of the DARPA GloMo Program. *IEEE Personal Communications*, 3(6):34–43, December 1996.

[18] David A. Maltz, Qifa Ke, and David B. Johnson. Emulation of Ad Hoc Networks. Talk delivered at the VINT Project Retreat, June 1999. Slides available from http://www.monarch.cs.cmu.edu/papers.html.

[19] Yukio Murayama and Suguru Yamaguchi. DBS: A Powerful Tool for TCP Performance Evaluations. In *SPIE Proceedings of Performance and Control of Network Systems*, November 1997.

[20] B. Noble, M. Satyanarayanan, D. Narayanan, J.E. Tilton, J. Flinn, and K. Walker. Agile Application-Aware Adaptation for Mobility. In *Proceedings of the 16th ACM Symposium on Operating System Principles*, pages 276–287, St. Malo, France, October 1997.

[21] Brian Noble, M. Satyanarayanan, Giao Nguyen, and Randy Katz. Trace-Based Mobile Network Emulation. In *Proceedings of SIGCOMM '97*, pages 51–61, September 1997.

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

[23] J. Postel. Transmission Control Protocol. RFC 793, September 1981.

[24] Neil Siegel, Dave Hall, Clint Walker, and Rene Rubio. The Tactical Internet Graybeard Panel Briefings. U.S. Army Digitization Office. Available at http://www.ado.army.mil/Briefings/Tact%20Internet/index.htm, October 1997.

[25] Bruce Tuch. Development of WaveLAN, an ISM Band Wireless LAN. *AT&T Technical Journal*, 72(4):27–33, July/August 1993.

[26] Gary R. Wright and W. Richard Stevens. *TCP/IP Illustrated, Volume 2: The Implementation*. Addison-Wesley, Reading, Massachusetts, 1995.